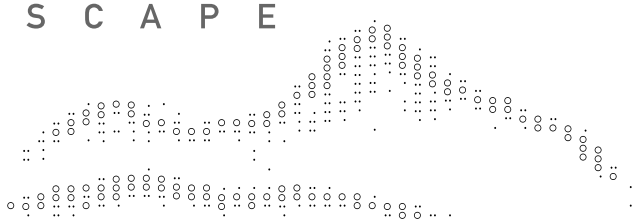
e S C A P E



The Electronic Landscape In Practice

Deliverable D4.4

Document ID	eSCAPE -D4.4
Status	Final
Type	Deliverable
Version	1.0
Date	August 2000
Editors	Tom Rodden and John Mariani
Task	4.1, 4.2, 4.3

© The eSCAPE Project, Esprit Long Term Research Project 25377

Project coordinator:

Tom Rodden
Computing Department
University of Lancaster
Lancaster LA1 4YR
United Kingdom
Phone: (+44) 524 593 823
Fax: (+44) 524 593 608
Email: tom@comp.lancs.ac.uk

The eSCAPE project comprises the following institutions:

Swedish Institute for Computer Science (SICS), Stockholm, Sweden
University of Lancaster, Lancaster, United Kingdom (Coordinating Partner)
University of Manchester, Manchester, United Kingdom
ZKM, Germany

Editors of this report:

Tom Rodden
Computing Department
Lancaster University
Lancaster
LA1 4YR
United Kingdom
Tel: +44 1524 593823
Fax: +44 1524 593608
Email: tom@comp.lancs.ac.uk

John Mariani
Computing Department
Lancaster University
Lancaster
LA1 4YR
United Kingdom
Tel: +44 1524
Fax: +44 1524 593608
Email: jam@comp.lancs.ac.uk

ISBN: 1-86220-098-X
Lancaster University, 2000
Report and further eSCAPE information available from http://escape.lancs.ac.uk/

Table of contents

INTRODUCTION AND OVERVIEW 1

The structure of the deliverable 2
Part I: The physical electronic landscape 3
Part II: The abstract electronic landscape 3

CHAPTER 1: LEARNING FROM THE TOURIST DEMONSTRATOR 7

CALIBRATING THE PHYSICAL ESCAPE DEMONSTRATOR 7
The need for a new landscape 10

CHAPTER 2: THE MIXING BAY STUDY - SPACE, PLACE AND INTERSUBJECTIVITY 15

THE SETTING 15
SPACE AND PLACE IN CSCW 23
THE MIXING BAY STUDY: SOME IMPLICATIONS FOR DESIGN 27

CHAPTER 3: THE DESIGN OF PLACEWORLD 31

THE USER INTERFACE AND PRESENTATION SYSTEM 32
THE VIRTUAL ENVIRONMENT OF PLACEWORLD 36
The content design of Placeworld 39
Navigation 39
Mapping, signposting, zoning, persistence, hyperlinks 40
Traces and other temporal markers 41
Importing and constructing your own places in Placeworld 42
Access models 42
Placing existing eSCAPE databases in Placeworld 42
Functionality of the touchscreen 44
Interpersonal contact and communication in Placeworld 45
The scalability of Placeworld 46

CHAPTER 4: STRUCTURE AND USER INTERFACE OF PLACEWORLD 47

ARCHITECTURE OF PLACEWORLD 47
Cylinders 48
Individual Cylinders 49
Generator Cylinders 50
Artwork Cylinders 51
"YOU" Cylinder 52
Paths 52
Avatars 53
Navigating and Interacting in the Placeworld 55
Communicating in the Placeworld 59
Controlling the Placeworld 60

CHAPTER 5: PLACEWORLD TECHNICAL REALISATION 63

CHAPTER 5A: ARCHITECTURE OF THE DEVA SYSTEM: ADDRESSING THE ISSUES OF 'SCALE' IN VIRTUAL REALITY AND ELECTRONIC LANDSCAPES 65

DISTRIBUTION AND COMMUNICATIONS 68
The distribution architecture 68
CREATING AND ADDRESSING ENTITIES 69
Entity migration 71
Server reconfiguration 71
Networking protocol 71
SYNCHRONISING THE BEHAVIOUR OF ENTITIES 71
Defining the behaviour of entities 73

Components ... *74*
Methods and Filters ... *75*

CHAPTER 5B: IMPLEMENTATION OF THE PLACEWORLD - MAVERIK CLASSES 77

AVATARS ... 78
Cylinders ... *79*
SMS SILHOUETTES ... 80
Signposts ... *82*
Image-Boards ... *83*
Replicator ... *84*
Lens-Flare ... *85*
Sky/Groundplane ... *86*
Icons ... *86*

CHAPTER 5C: THE PLACEWORLD ACCESS MODEL ... 89

THE ACCESS MODEL ... 90
Detailed anatomy of a Key ... *92*
UserRequest ... *94*
PollBooth ... *94*
Group ... *94*
PRESENTING THE MODEL TO USERS ... 95
Representing users relation to the environment ... *96*
Representing permissions in the environment ... *97*
Zone Permissions ... 97
Entity Permissions ... 97
Describing permissions in the environment ... *98*
The Zone access table ... 98

CHAPTER 6: THE ABSTRACT ESCAPE REVISITED ... 103

THE ISSUES: DESCRIPTIONS AND SOLUTIONS ... 103
Reliability ... *103*
SCALABILITY ... 104
FULL TITLE NOT AVAILABLE ... 105
INCONSISTENCY OF COLOUR ... 106
INDIVIDUAL ITEMS AND CLASSMARK RELATIONS ... 107
CENTRED VIEW ... 107
CLICKING ON OBJECTS WITHIN A CLOUD ... 108
OVERLAPPING OBJECTS ... 108
RELATED KEYWORDS ... 108
DIFFICULTY OVER THE CONCEPT OF WEIGHTS ... 109
SUPPORTING SEARCHING BY ELIMINATION ... 109
RELATING SEPARATE SEARCHES ... 109
POSITIONING OF OBJECT TEXT IN SPACE ... 110
RETURNING TO AN OBJECT ... 110
SUMMARY ... 111

CHAPTER 7: THE LIBRARY FRONT END -TECHNICAL REALISATION ... 113

AIMS ... 113
THE ENVIRONMENT ... 114
Architectural bottlenecks ... *116*
THE Q-PIT PROCESS REVISITED ... 116
ALGORITHMIC BOTTLENECKS ... 117
Similarity computation ... *118*
Interacting with the database ... *118*
Force placement algorithm ... *119*
Hull generation ... *121*
MOVING TOWARDS MULTI-USER Q-PIT ... 124
A brief overview of Deva ... *124*

THE Q-PIT IN ACTION ... 126
Selecting an object ... *129*
Bookmark ... *131*
Delete ... *131*
More forms of selection ... *132*
Selecting hulls ... 132
Multiple selection ... 133
Reconfiguration of the molecule using interest weights ... *133*
Manual control of the reconfiguration ... 133
Halting and restarting the force placement algorithm ... 133
Forcing reconfiguration of the space ... 134
Finding similar books ... *134*
History Trails ... *134*
ADDRESSING THE ISSUES FROM THE FIRST SET OF STUDIES ... 135

CHAPTER 8: CALIBRATING THE LIBRARY DEMONSTRATOR ... 139

THE LIBRARY DEMONSTRATOR IN USE ... 139
EMERGENT FEATURES AND ISSUES FOR DESIGN ... 145
Sequence 1. ... *145*
Sequence 2. ... *147*
OUTCOMES ... 148
FUTURE WORK ... 148

CHAPTER 9: THE DESIGN OF THE ABSTRACT ESCAPE INTERFACE ... 159

History Menu ... *160*
Search Menu ... *161*
Information option ... 161
Keyword option ... 162
Weighting option ... 163
Clipboard option ... 164
Delete option ... 165
Navigation in the Q-PIT ... *166*

CHAPTER 10: DEVELOPING A GENERIC ARCHITECTURE ... 167

THE EXISTING LIBRARY DEMONSTRATOR ARCHITECTURE ... 167
Components of the Year 2 Abstract Landscape Demonstrator Architecture ... *169*
OPAC : On-line Public Access Catalogue ... 169
Data Extractor ... 169
MySQL Database and JDBC ... 169
Q-PIT and DIVE ... 169
The Architecture as a Whole ... *171*
Generating the Q-Space ... *171*
Populating the repository ... *171*
Producing the Q-Space ... *172*
Generation of a Benediktine space ... 172
Generation of a Similarity Matrix ... 173
Generation of a Minimum Spanning Tree ... 173
Identify regions within the tree ... 173
Applying a force displacement algorithm ... 174
Generation of data clouds ... 174
Further interactions ... *174*
Factors: Weighting and Exactness ... 174
CONCLUSIONS ... 175
Issues raised in user studies ... *176*
Supporting Abstract Landscapes in Future Work ... *179*
Conclusions ... *181*
REQUIREMENTS FOR A NEW Q-PIT ARCHITECTURE ... 182
The New Q-PIT Architecture ... *182*
Data Source ... 183
Data Extractor ... 183

MySQL Database and JDBC ... 185
Q-PIT Server ... 186
Q-PIT ... 186
The New Q-Space generation process ... *187*
Connecting to the Q-PIT server: The Q-Space API ... 187
Handshake: supplying the user name and client type ... 189
Listing Data Sources ... 190
Selecting the Data Source ... 190
The Keyword Search ... 191
Collection of data ... 191
Q-PIT processing ... 192
Sending the initial Q-space ... 192
Updating Q-space ... 194
Creating another query across the same socket ... 195
Changing the space parameters ... 195
Removing object, groups and Q-Spaces ... 197
IMPLEMENTATION ISSUES ... 198
CONCLUSIONS ... 199
REFERENCES ... 201

CHAPTER 11: ALTERNATIVE INTERFACES ... 203

Chapter 11a : The Tiltboard ... 203
Chapter 11b: The Pond ... 203
Chapter 11c: The WAP interface ... 203

CHAPTER 11A: ALTERNATIVE INTERFACES - THE TILTBOARD ... 205

TECHNICAL DESCRIPTION ... 206
Functionality ... *207*
GETTING 'HANDS ON' THE PROTOTYPE – THE OUTCOMES OF USER EXPERIENCES ... 208
Improvements implemented following user trial ... *210*
1. FREEZE MECHANISM ... 211
2. ALTERNATE MODE OF NAVIGATION ... 212
3. NAVIGATION BUTTONS ... 212
4. RETURN BUTTON ... 212
5. SWITCHING BETWEEN MODES ... 212
CONCLUSION ... 213

CHAPTER 11B: ALTERNATIVE INTERFACES - THE POND ... 215

THE POND AND THE LIBRARY DEMONSTRATOR ... 215
The content ... *215*
The set-up ... *216*
The ecosystem ... *217*
Behaviour and interactions ... *217*
Pressure pads and RFID tags ... *218*
The Sonic Interface ... *220*
Real Time Video Tracking ... *220*
Shoulder to shoulder collaboration ... *221*
Web connection ... *221*
JIVE ... *221*
DISCUSSION ... 222

CHAPTER 11C: ALTERNATIVE INTERFACES - WAP ... 225

WML – A BRIEF DESCRIPTION ... 225
Q-Spaces and WML ... *226*
The Architecture ... *226*
Components of the Alternative Visualiser ... *227*
Q-PIT: the Q-Space Generator ... 227
Cocoon: a publishing system ... 227
WAP-enabled device ... 228

ACCESSING A Q-SPACE 229
Keeping the User Informed *231*
The Q-Space in XML *232*
Displaying the Results *234*
Changing Q-Space Parameters *237*
IMPLEMENTATION ISSUES 238
Sessions *238*
Connection versus Connectionless *239*
CONCLUSION 240

CONCLUDING REMARKS 243

Introduction and Overview

Tom Rodden
Lancaster University

From the outset the eSCAPE project has sought to develop an electronic landscape. We clearly stated this desire in the overall objective of the project as expressed in the original project programme.

> "The eSCAPE project will investigate the exploration and development of the concept of an 'electronic landscape' (e-scape) as a virtual environment which provides interconnections to other virtual environments. An e-scape is a place where places meet and where all those places are inhabited information spaces which can contain representations of persons (avatars), objects/information and artificial agents. An e-scape supports the integration and coexistence of multiple shared virtual environments." (ESCAPE project programme, pp 3)

The work, during this final year of the eSCAPE project, has focused on the development of a final demonstration electronic landscape that both realises this original vision and builds upon the work of the second year of the project.

The first two years of the project saw the investigation and clear articulation of the notion of an electronic landscape and its exploration through collaborative development with real world users. In the second year of the project two distinct styles of electronic landscape emerged; the physical escape and the abstract escape. Demonstrators of both these styles of eSCAPE were constructed in partnership with real world users and assessed in practice.

This deliverable describes the final year work of the eSCAPE project. It is concerned with the development of an electronic landscape that builds upon last years work on the development of a physical and abstract electronic landscape. The development of the single electronic landscape presented in this deliverable has been undertaken as a means of consolidating and integrating the work of the project and presenting the work in a coherent manner. A number of objectives drove the work of the demonstrator reported:

- The realisation of an electronic landscape that allows a number of different electronic environments to be brought together.
- The development of a demonstrator that presents the developed environment using novel interactive techniques in order to provide a new interactive experience
- The bringing together of the work on social scientific studies, the different artistic explorations and the development of new environments.
- The involvement of real world users in the development of the electronic landscape in order to meet the demands of a particular community.

The electronic landscape development reported in this deliverable, builds upon and converges the work on the physical and electronic landscapes of previous years. The work on the physical landscape has been significantly extended from last year to provide an environment that can be more readily inhabited by users

and amended. The physical environment also provides a point of integration with other electronic landscapes including the abstract electronic landscape.

The evolution of the physical electronic landscape from a single application to support the work of the tourist center to a more generic electronic infrastructure that allows a range of different styles of electronic environment to coexist has been driven by a reassessment of the experiences of last years demonstrator. The work has reflected on the results of a number of ethnographic studies and has chosen to develop a design driven electronic landscape. The resultant environment called Placeworld is presented in terms using a novel interactive configuration and builds upon previous work by Jeffrey Shaw on Place-a user manual.

The development of this single illustrative electronic landscape as a means of presenting the work of the project also offers us a distinctive dissemination vehicle that is designed to be presented to and experienced by the general public. The aim is to allow this demonstrator to be presented across a range of museums across Europe as a means of disseminating the work of the project. This will be complemented by the free dissemination of this software designed to run on standard workstations.

The physical landscape allows access to a range of environments including the extended abstract electronic landscape that allows access to a range of information sources. This environment has been further refined and developed in partnership with a user community and supports resource discovery in on-line information. In addition to developing the functionality of this electronic landscape, a novel interface has been designed with a similar style to that provided by Placeworld. This similar interface seeks to make the transition from the physical to the abstract aspects of the developed electronic environment as smooth as possible.

The work of the abstract electronic landscape has also supported the development of a generic architecture. This supports the generation of electronic landscapes from a large number of sources and allows these to be experienced via a diverse set of interfaces. This architecture has been exercised by an exploration of a radical set of interfaces to the shared pool of common information.

The development of the different electronic landscapes presented in this deliverable also represent concrete instantiations of the common method described in Deliverable D4.3. This shows the application of the iterative process of articulating new research concepts through artistic explorations, grounding these concepts with understandings of the real world, developing these concepts through the construction of a demonstrator and the calibration of these demonstrators to inform the refinement of the concepts.

The structure of the deliverable

This deliverable offers a broad account of how the project has brought the work of the project together and how it has advanced the demonstrations of the second year. The deliverable is divided into two distinct components:

Part I: The physical electronic landscape

This section of the deliverable focuses on the evolution of the physical demonstrator into the final integrated Placeworld electronic landscape.

Chapter One presents a brief review of the experiences of the physical electronic landscape from the second year and the underlying motivation for the decision to evolve the physical demonstrator to form the Placeworld demonstrator.

Chapter Two presents an ethnographic study of how users exploit the various affordances of a shared public environment and the activities that need to be supported in an equivalent environment. The focus of the study is on the nature of space and how this is configured and amended by its inhabitants to make it into a particular place

Chapter Three presents the design of Placeworld. This was an artistic led design informed and supported by the experiences of last year and the on-going studies reported in chapter 2. The environment developed builds upon the development of the physical demonstrator last year and previous work on the construction of an installation called Place-a user manual.

Chapter Four described the development and design of the user interface provided to Placeworld and how an interface has been designed to meet the demands of the specific arrangement of interactive devices. The designed interface also seeks to provide a familiar feel with abstract demonstrator that is embedded within the Placeworld.

Chapter Five provides an overview of the novel technologies used to construct the Placeworld demonstrator. This chapter is divided into three related subchapters covering the supporting architecture, the particular graphics techniques exploited and the access model used to manage the environment.

Part II: The abstract electronic landscape

This section of the deliverable focus on the continued iterative development of the abstract electronic landscape and its placing within the Placeworld to demonstrate the final vision of a unified electronic landscape.

Chapter Six revisits the experiences of the abstract electronic landscape from last year and the lessons learned for the further refinement this year. This chapter reviews the lessons learned from the calibration of the initial version of the demonstrator and the refinements required.

Chapter Seven presents a refined version of the abstract electronic landscape demonstrator. This refined version responds to the issues raised from the previous year of the project and introduces a series of amendments to support the work of the real world community.

Chapter Eight presents a further situated evaluation of the amended interface and considers the refinement in terms of the demands of users to find particular on-line resources. The amendments suggested are responded to in a further refinement of the interface.

Chapter Nine describes the redesign of the abstract electronic landscape to provide a similar feel to the Placeworld electronic landscape demonstrator. This redesign build upon some of the issues raised in chapter seven while also exploring how to reduce the impedance involved in shifting from one style of electronic landscape to the other within the Placeworld demonstrator.

Chapter Ten presents the development of a generic abstract electronic landscape engine as part of the generalisation of the work on the development and use of the abstract electronic landscape this generic architecture allows the presentation of multiple different information sources across a diverse collection of devices.

Chapter Eleven presents an exploration of a series of alternative interfaces to the abstract electronic landscape. These are presented in a series of subchapters exploring the development of a novel "tilt & touch" device, the construction of an electronic information environment and the development of a mobile interface using WAP technology.

Part I: The Physical Electronic Landscape

Chapter 1: Learning from the tourist demonstrator

Andy Crabtree, Steve Pettifer, Tom Rodden
Lancaster University, The University of Manchester

The work on the physical electronic landscape reported in this deliverable builds upon the experiences and lessons gained from the second year work on the tourist physical escape demonstrator and an analysis of its use in practice by a community of users within the northwest of England. The tourist demonstrator exploited a number of research concepts articulated through a series of multimedia art installations. The tourist demonstrator provided a grounding of these "blue sky" research concepts in conjunction with a real community of users. The main aim of the demonstrator was to uncover users' reactions to the nature of a physical electronic landscape and to better understand the affordances of this form of landscape.

The tourist information landscape provided a concrete demonstration of the ways in which the different approaches of computer scientists, artists and sociologists could be combined to meet practical ends. In this regard the investigation of the properties of physical escapes followed the broad methodological approach outlined in deliverable D4.3. We began by using a series of interactive art pieces as a means of *generating "blue sky" concepts*. These concepts were then *grounded* by exploiting the emerging results from an ethnographic study of users of a tourist information center. These concepts were *configured* in the construction of a demonstrator and they were finally *calibrated* through a series of situated evaluations.

This chapter briefly reviews the calibration of the physical eSCAPE demonstrator and the consequences to emerge for the further development of the work during the final year of the project. In particular, we consider the reasons for developing the integrated Placeworld demonstrator and the need to undertake additional studies that ground the concepts to emerge from the artistic investigations.

Calibrating the physical eSCAPE demonstrator

The calibration of the tourist electronic landscape undertaken at the end of the second year and the start of the final year of the projects work, highlighted a major issue in the development of this form of landscape within the context of its use in the tourist information center. The physical escape aims to support the structuring and understanding of an electronic landscape. While this can be grounded by a consideration of the activities that take place within the tourist information center, the nature of the physical escape is that it is a framework for activity. This is analogous to the World Wide Web, which provides an

overarching framework to support the development of a diverse set of activities rather than supporting a single focused application.

The vision of the Physical electronic landscape is that, in the same way that a real world city acts as a backdrop for a diverse set of activities the physical electronic landscape should provide a place in which different activities take place. One result of the broad intent of the physical electronic landscape is that a mismatch existed between the research aims of the development of the landscape and the setting chosen to ground these concepts. Essentially, the needs of the setting was to have a tourist information resource that provided information about the local area and the needs of the project to understand the nature of the physical electronic landscape. Our fundamental concern was that we were starting to build an on-line tourist information guide rather than undertake research on future information landscapes. A series of situated assessments undertaken at the start of the third year confirmed this and left the project with a clear decision. Should it develop an on-line tourist guide for a particular location or should it develop a more generic demonstrator that illustrated the lessons learned from the tourist demonstrator? Given the considerable level of activity in electronic tourist guides we eventually decided to revise the development of the physical electronic landscape in order to align it with the need to develop a general landscape of activity of utility to a broad population.

The situated evaluations formed part of the calibration phase within the common methodology we outline in D4.3 for this demonstrator. As well as raising issues about the nature of the demonstrator it also highlighted the need to revisit the artistic demonstrators that provided our source of "blue sky" concepts and to reconsider the structure and nature of the electronic landscape itself.

The calibration of the electronic landscape highlighted a major misconception of the nature of the environment. The assumption commonly felt by users and those exposed to the physical demonstrator was that it was configured to support the work of a Tourist Information Centre (TIC).[1] The demonstrator did not do so and was, quite naturally, found wanting. Supporting the work of the TIC was never the intention in undertaking design however. Rather, the intention was to *learn lessons from* a real-world situation that paralleled a conceivable situation that users would likely experience in interacting with a virtual cityscape in a projected future. It was reasoned that the real-world tourist is, in fundamental ways, confronted with a place that they do not know well in which they wish to find something, or some things, of interest within it of direct relevance to their personal concerns and curiosities. It was reasoned that the virtual tourist will find him or herself in a parallel situation in an electronic landscape and will be faced with the irremediably practical task of “finding something interesting to do” from amongst a potentially massive range of activities available online. The point and purpose of the ethnographic studies of the TIC was, then, to inform development

[1] West, A. & Pettifer, S. (eds.) (1999) *The Tourist Physical Electronic Landscape Demonstrator*, eSCAPE Deliverable 4.2, Esprit Long Term Research Project 25377, ISBN 1 86220 080 7.

as to the co-operative work-practices which underpin the identification of "something interesting" and use such understandings as *resources for thinking about organising* virtual environments to meet this abiding practical concern.

The ethnographic study identified an ensemble of ordinary categorisation practices that people engage in for purposes of 'working up' things of potential interest in the course of establishing a specific sense of "what can we do here?" - categorisation practices which flowed seamlessly into the work of formulating directions to places identified as "interesting".[1] Our understandings of these work-practices were employed as resources for thinking about design and, and as such, informed the formulation of a design scenario that posited a city-like eSCAPE that utilised art concepts (notably PLACE)[2] and supported categorisation work through the deployment of a topographical structure that 'deformed' in response to user queries.[3] A version of the posited application was then developed,[4] and subject to user trials in order that the organising design concepts might be confronted (and thus elaborated) by *some* practical circumstances of use and design 'calibrated' accordingly.[5] In the course of that confrontation a particular 'problem' emerged that had fundamental consequences and shaped the nature of subsequent work undertaken this year.[6]

> The addition of new data and the modification of the environment represents a real challenge for the environment in that the situated nature of the application raises real questions as to who owns the environment and how this environment is modified for its general inhabitants. The current environment is constructed from a database of resources that is linked to a well-specified ground plane. Addition to the environment is done through the database and developers can access this database in order to develop this environment. Currently the addition of new resources focuses on technical developers of the environment and no higher level tools are provided to support the development and construction of the environment. This raises the direct

---

[1] Crabtree, A. (1999) *The Tourist Information Centre Study*, The Tourist Physical Electronic Landscape Demonstrator (eds. West, A. & Pettifer, S.), eSCAPE Deliverable 4.2, p. 3-12, Esprit Long Term Research Project 25377, ISBN 1 86220 080 7.

[2] Shaw, J. & West, A. (1999) The Supporting Artworks, The Tourist Physical Electronic Landscape Demonstrator (eds. West, A. & Pettifer, S.), eSCAPE Deliverable 4.2, p. 29-38, Esprit Long Term Research Project 25377, ISBN 1 86220 080 7.

[3] Crabtree, A. & Rodden, T. (1999) *From ethnographic study to systems design: formulating design proposals for the physical electronic landscape*, The Tourist Physical Electronic Landscape Demonstrator (eds. West, A. & Pettifer, S.), eSCAPE Deliverable 4.2, p. 13-28, Esprit Long Term Research Project 25377, ISBN 1 86220 080 7.

[4] Rodden, T., Trevor, J., Mariani, J., West, A., Pettifer, S., Cook, J. (1999) *The Tourist Information Centre Electronic Landscape*, The Tourist Physical Electronic Landscape Demonstrator (eds. West, A. & Pettifer, S.), eSCAPE Deliverable 4.2, p. 69-86, Esprit Long Term Research Project 25377, ISBN 1 86220 080 7.

[5] Murray, C., Arnold, P., West, A. (1999) *The Tourist Information Centre in Use*, The Tourist Physical Electronic Landscape Demonstrator (eds. West, A. & Pettifer, S.), eSCAPE Deliverable 4.2, p. 87-110, Esprit Long Term Research Project 25377, ISBN 1 86220 080 7.

[6] West, A., Pettifer, S., Cook, J., Rodden, T. (1999) *Concluding Remarks and Future Possibilities*, The Tourist Physical Electronic Landscape Demonstrator (eds. West, A. & Pettifer, S.), eSCAPE Deliverable 4.2, p. 111-115, Esprit Long Term Research Project 25377, ISBN 1 86220 080 7.

> question of *who* owns the environment and *how* is its modification and amendment managed? (West *et al.* 1999, p. 112 emphasis added)

The need for a new landscape

The 'problem', as it were, was not so much that the TIC environment was under populated (which it was) but that the designers 'owned' the TIC demonstrator, not so much in the sense of it being their property but rather, in the sense that they owned it for all practical purposes beyond navigation. Users could not make changes to the environment, could not manage modifications and amendments – any such work was the technicians' concern. Consequently, the environment was practicably unusable. It couldn't be populated by users, updated, and otherwise be maintained and the simple fact that users could not alter the fabric of space presented a serious obstacle to the potential uptake and use of such electronic landscapes. If electronic landscapes of this kind are to entertain any prospect of widespread use it was reasoned that users need to 'own' the environment, for as the user trials made it very clear, it is practically impossible to construct *prefigured* yet dynamic environments of any lasting utility. In light of user experiences evoked in confronting the TIC environment with practical circumstances of use, the design-problem became one concerned with 'how' to provide users with 'ownership' of the physical environment.

The *prima facie* 'solution' to this design problem is to let users construct their own environments. But what would this entail? What kind of mechanisms would we need to provide in order to allow users to become owners of the physical eSCAPE? Again, rather than theorise the matter, we sought *to consult* the real world as a resource for thinking about design. We wished to realise a second generation of the physical demonstrator that was owned and managed by users. In order to do so we needed to undertake two things in line with the methodology outlined in D4.3:

- We needed to generate new "blue sky" concepts inspired from an art and design traditions tradition.
- We needed to ground these concepts the emergent research concepts by placing them alongside understandings of real world activities.

In order to do so we chose to undertake the design led development of a physical electronic landscape. We also decided to explore the different ways in which alternative electronic landscapes can be brought together. We wished this landscape to be real and evolving and to be controlled by its own inhabitants. This remit underpinned the design led development of Placeworld described in this section of the deliverable.

We also wished to ground this development through reference to real world settings and real world activities. Given that we are concerned to develop environments of public utility this led us to investigate the social organisation of a public space where on-line resources (including the abstract demonstrator) were

accessed and used and resulted in the study of the mixing bay reported in Chapter 2. The purpose of the study was to develop an understanding of some of the ordinary ways in which public places are socially constructed in the practical activities of inhabitants. The outcome of the study was the notion of the construction of familiar 'structures' of practical action. These 'structures' consist of familiar representations of place which are assembled, managed, and assigned uses by users in constructing virtual spaces of practical utility. The use of familiar representations provides a primary means of engendering intersubjective awareness and co-ordination of the space's work in and through the practised use of familiar representations as publicly available 'equipment' for conducting activities. Thus, and for example, users can create places that consist of 'streets' and street furniture such as 'signs' and 'billboards', and 'buildings' and office furniture such as 'shelves' and 'filing cabinets'. In considering the real world character of representations, the mixing bay taught us that places consist of public and private aspects: public spaces also consist of private spaces. Thus, and again for example, in constructing a place consisting of 'buildings' or 'offices in buildings', some will be 'open' to anyone and others accessible to only a select few, even just one person.

In using the mixing bay study as a resource for thinking about design it became apparent that we would need to develop an object library providing (and providing for the import of) representations out of which *users can construct places of their own*. The mixing bay study also suggested that the representations comprising the user-constructed landscape would require support for the 'invisible' work implicated in their practical management. Work such as updating the information contained in particular representations, adding new representations, and in making changes to the fabric of the overall space (or sections of). This would require the development of an editor. The public/private character of real world places also suggested the need for an access model enabling users to grant rights and privileges both for using particular representations and editing them. Consulting the real world, real time social organisation of space thus inspired the development of an integrated software package providing support for the import and storage of representations, the editing of representations comprising the user-constructed space, and access control. This arrangement allows users, in principle at least (we recognise that a great deal of 'calibration' work with users is required to move beyond 'in principle'), to create shared worlds from familiar representations that 'work' in some very familiar ways. These, thus foster, the learnability of electronic landscapes in drawing on 'what anyone knows' about the places in which they conduct their daily affairs.

Significantly, the development of Placeworld has seen the migration through 'configuration' of artistic concepts (reported in this section in chapter 3) to meet practical circumstances of use. This has required the development of a set of supporting mechanisms (reported in chapter 4) that allow the editing and control of the world. The Placeworld builds upon the physical demonstrator from last year and the previous work on "Place - a user manual" by developing an environment

that exploits a landscape of cylinders that allow the configuration of and access to a series of thematic places.

In addition to contributing directly to a tangible success in the design of a concrete artefact, the migration of Place represents a concrete success for the common methodology, marrying the work of artists and ethnographers together with technological design. One of the central (albeit emergent) goals of the project has been to develop a methodology for different specialists to work together. The development of the physical eSCAPE again demonstrates that the selection of working partners and the approach to co-design described in Deliverable 4.3 produces worthwhile and meaningful results. The development and refinement of a common research methodology is clearly an extremely important contribution to the wider development community and this year's work has seen the integration of art, ethnography and technology development better reflected in specific designs.

Given our decision to embark for the final year on the construction of a more generic and integrative demonstrator it might be asked: what of the TIC demonstrator? It may appear to have disappeared and no longer be of concern. This is not the case however. As discussed above, the TIC demonstrator was never intended to support TIC work but was, instead, intended as a resource for thinking about the design of physical eSCAPEs and confronting subsequent design artefacts with practical circumstances of use in order that development might be usefully and constructively informed by end-users. *This* goal – the specified goal of the TIC demonstrator - was achieved and a course of design responding to user requirements was undertaken in this final year resulting in the Placeworld. This design addresses the central problems encountered by users in confronting the TIC demonstrator and enables end-users to construct and maintain their own environments.

Placeworld also allows the project to end with a vision of the future that embodies the original aims of the project

Imagine a situation in which a multiplicity of Placeworlds exist – hundreds if not thousands, even tens of thousands of globally distributed and different Placeworlds available online. A fundamental concern is how users find something of interest from amongst all those differing environments? In considering that issue, we are back where we began in year two of the project. This time, however, we have developed the tools whereby physical electronic landscapes may be created, connected, and queried. Naturally, a great deal of practical work needs to be accomplished in order to fully realise this vision, work that we have not been able to accomplish this year but then, that is a matter of what is practically achievable within the time-frame available. Nonetheless, one can easily *envision* the situation described above and see the relevance of the TIC demonstrator to it.

In the rest of this section of the deliverable we shall describe the development of Placeworld as an integrated physical electronic landscape. In the following chapter we start by presenting an ethnographic study of a public space that is managed by its inhabitants to support a diverse range of activities.

Chapter 2: The Mixing Bay Study -
Space, Place and Intersubjectivity

Andy Crabtree, Steve Pettifer
Lancaster University,The University of Manchester

The motivation behind the following ‘quick and dirty’ study is to ground the "blue sky" research concepts underpinning the development of the Placeworld demonstrator. The study complemented the generation of new (and often radical) concepts from art and design traditions with an understanding the everyday activities of real world users. A principle thrust of the study was a need to develop an awareness of intersubjectivity *as an endogenous organisational feature of* public spaces towards the end of formulating design proposals for the development of facsimile electronic landscapes available to public use. That is, towards the development of virtual worlds closely resembling physical spaces and places of public utility. By ‘intersubjectivity’ werefer to the embodied *social* use and reflexive organisation of physical space and its material arrangements or composite equipment. The focus of the study, then, is on the activities or ‘lived work’ that take place within public places – in this case, the ‘mixing bay’ in a sociology department - and the ‘equipment’ that is used in the course of accomplishing that work. Naturally, not all public spaces are the same. Very different kinds of work take place in parks, shopping precincts, at sports stadiums, bus stations, on highways, and the rest, and each distinct space and place consists of different and unique material arrangements (although some similarities, such as the use of signs for example, may exist). Despite the heterogeneous character of public spaces and places the study may go some way to elaborating some (and only some) trans-situational features that go towards making up the very ‘publicness’ of public spaces and places. The study is in no way intended to be exhaustive but a preliminary exploration of the public character of public spaces; one intended to sensitise us to that public character, or aspects of it at least, and inform us as to potential avenues of realisable exploration in design.

The setting

The setting within which activities take place is, naturally, a prominent feature in understanding the social organisation of space and place. This is particularly evident in the case of the mixing bay where the material arrangements of the room (its equipment) are ‘laid out’ with a view to making information and resources publicly available. The layout of the space consists of distinct arrangements of equipment that comprise the public character of the space and afford its distinct ‘placeness’. That is, its uniqueness as ‘just this’ place: a sports stadium, a shopping precinct, a mixing bay in a sociology department. Hughes *et al.* (1994: 157) suggest that the layout of the setting constitutes an ‘ecology of awareness’ important to the development of CSCW systems. The purpose in this section of

the report is to explicate that ecology and identify important aspects of its organisation.

The mixing bay is situated in the Sociology Department at Lancaster University (UK). It is a space available to public use by staff, students, and others. It houses a comfortable seating area, refreshment facilities, various artefacts serving organisational purposes, and a public workstation. Figure 1. below portrays the ecology of the space and its equipment.

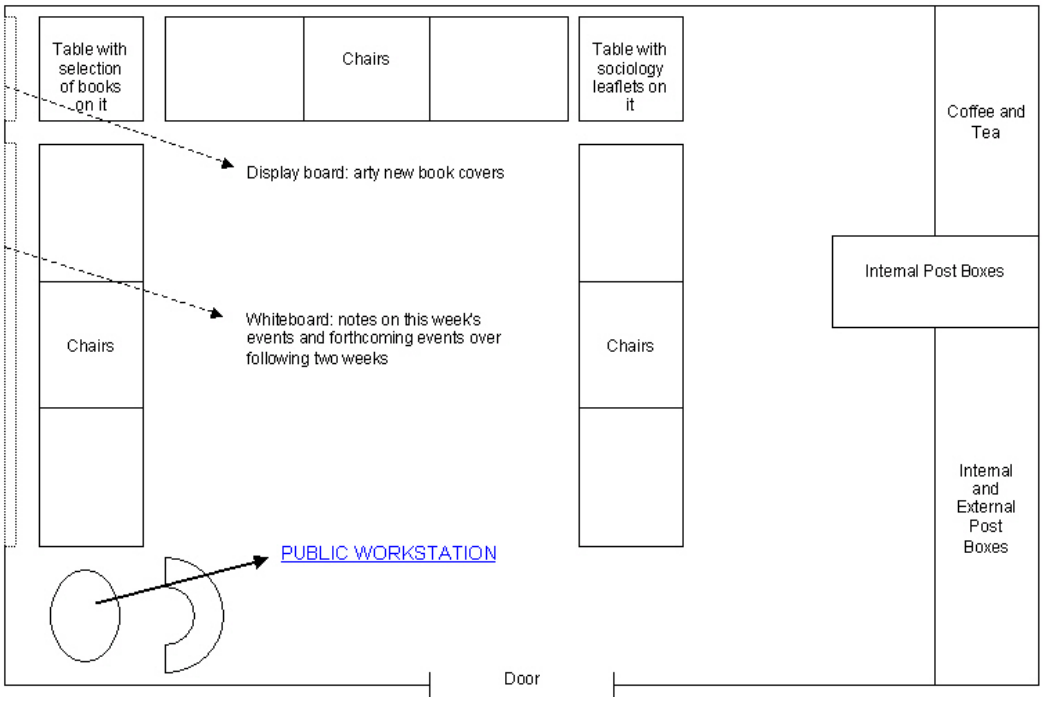


Table with selection of books on it
Chairs
Table with sociology leaflets on it
Coffee and Tea
Display board: arty new book covers
Internal Post Boxes
Chairs
Whiteboard: notes on this week's events and forthcoming events over following two weeks
Chairs
Internal and External Post Boxes
PUBLIC WORKSTATION
Door

Figure 1: The Layout of the Mixing Bay

The ecology of the space is made up of the ecology of its material arrangements, of each piece of equipment comprising the space. Which is to say, that *each piece* of equipment has its own publicly available organisation which goes to make up the public character of the space as a distinct whole. Thus, the whiteboard is a socially organised piece of equipment designed to communicate and thereby furnish an awareness of local activities and events to the space's population. Activities such as the topic, date, time and location of seminars, reading groups, and meetings, or events such as open days, guest lectures, or some important inter-personal occasion for the members of the space's population (such as the birth of child, for example). There is *a mundane orderliness* to equipment which constitutes and provides for its public availability, for its sociality and use in social activities. Thus, the whiteboard displays its contents in particular ways which provide for the public intelligibility of its contents. It is organised from left to right and from top to bottom as people naturally read in English. It displays its contents in an order of relevance according to the sequential progression of reading (from left to right and top to bottom). Consequently, imminent events are

displayed first, on the left at the top, according to natural methods of reading English. As reading progresses (from left to right and top to bottom) then so too does the temporal order of events projected on and by the whiteboard (which in this case cover a period of three weeks).

Underpinning the mundane but nonetheless essential orderliness of the whiteboard is *an organisation of 'invisible' work*. That is, work not seen at the whiteboard, or only fleetingly, but which the social use of the whiteboard *relies upon*. The contents of the whiteboard do not put themselves there but are placed there. Who does that work and what does it consist of? Well importantly, although anyone could put messages on the whiteboard, not anyone does. There is a background expectancy at work amongst the space's population that the whiteboard will display *organisationally relevant* information. Thus, the space's population do not place personal announcements (etc.) on the whiteboard. Indeed the vast majority of the space's population do not place anything at all on the whiteboard. The work of the whiteboard is accomplished, and the equipment practically managed in the course of that work, by a *select group* of the space's population, namely the department's support staff.

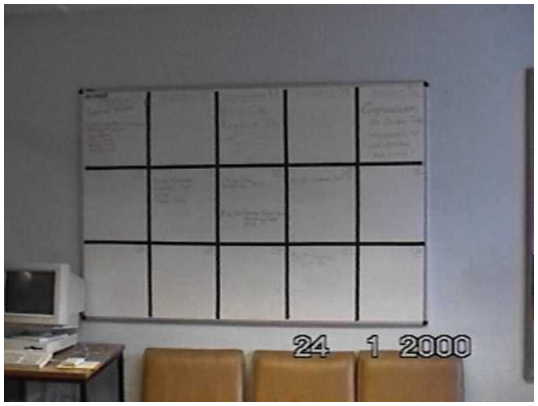


Figure 2: The Whiteboard

Each of the five members of the department's support staff occupy different organisational positions and carry out different organisational jobs of work. This organisation of activities does not carry through to the practical administration of the whiteboard however. Instead, and on a contingent 'got anything to put on the board?' basis, support staff co-ordinate their schedules to see what needs to go up on the whiteboard. The work of co-ordination trades on the use of diaries, both paper and electronic, email, and personal memory. Notably, support staff's diaries are not shared but private. In terms of the whiteboard's use, trivially it might be said that people come and read it. More importantly, however, the awareness of local activities the whiteboard promotes is supplemented with and *supported by* the use of email amongst the department's academic, support, and research staff. As contingencies dictate (such as the rescheduling of seminar or the cancelling of a guest lecture, for example) *changes* to the whiteboard are made public to the site's local staff through the use of email.

Just as the whiteboard is a socially organised piece of equipment which goes to make up the distinct 'placeness' of the space *in its orderliness, practical management, and use*, then so too are the other material arrangements which comprise the public character of the space. Namely, the post boxes, seating area, and refreshments area. Like the whiteboard, the post boxes are mundanely ordered pieces of equipment. They are ordered in *categories* of 'academic and support staff', 'research staff', 'post-graduates', 'internal' and 'external' post. The first three categories are not explicit, not displayed on the boxes themselves, but implicit and taken for granted amongst the space's population. Academic and support staff are *grouped together* in one area consisting of separate boxes (or 'pigeon holes') *by name* in *alphabetical* order. Research staff and post-graduates are also grouped together, research staff by name and alphabetical order, post-graduates by alphabetical order only (e.g. A-C, D-F, G-K, etc.). The ordering of post-graduate mail is, implicitly, by first character of last name. Internal and external post boxes (one of each) serve all user categories and, as pieces of equipment, are not ordered in any finer detail.

Figure 3: Post Boxes

As with the whiteboard, the use of the post boxes relies on a host of 'invisible' work. Two members of the support staff *divide the daily work* of 'sorting' between them. One sorts incoming mail in the morning and the other, outgoing mail – both internal, to other places within the university, and external, to places beyond the university – in the afternoon. The sorting of mail consists of *collaboration* with college porters who receive both internal and external incoming mail. The porters sort both sets of mail into departmental lots which are collected by support staff. Collaboration here occasionally consists in resolving practical troubles of identity. A practical problem may be one of establishing who a particular piece of mail is for? Is this mail for a member of the space's

population? Such troubles are more often than not resolved through support staffs' working knowledge of who comprises the space's population – i.e. a familiarity with the space's members. Where working knowledge runs out, the mail in question is *'passed on'* or *'redirected'* to staff at University House, where identifying the recipient becomes another's job of work. Mail from abroad also gives rise to troubles on occasion insofar as recipient names are written in a native language. For example, mail from Asia being particularly troublesome. A first course of resolution in such cases consists of checking personnel registries to see if a signature can be found that corresponds to the written name. Failing that, the case is redirected to University House. Once incoming mail has been collected from the porters, and any identity troubles resolved, the mail is sorted or placed in the recipient's pigeon hole. It may be collected almost immediately or stay their for several weeks. If mail is not collected after several weeks, staff seek to establish why, saving, redirecting, forwarding, or trashing mail in accordance with inquiries.

The orderliness of outgoing mail consists in sorting internal and external mail. Internal mail is *sorted into bundles* by college and the contents of each bundle secured with a rubber band. External mail is sorted into different classes (1st and 2nd class) stamped, airmail, and franked. 1st and 2nd class mail are sorted, bundled, and secured like internal mail. Airmail is placed in a white bag. Franked mail is placed in a brown bag. Importantly, the sorting of external mail is carried out with an eye towards identifying inappropriate use of mail facilities. All external mail within the space is paid for by the department on the basis that it is connected to organisational business. In the course of sorting external mail, that *mail is checked* then with an eye towards 'spotting anything suspicious', such as the posting of personal bills or private letters. Anything that 'looks suspicious' may be opened to confirm that suspicion, one way or the other, and if the suspicion proved correct, the 'culprit' brought to account. Once sorted, internal and external mail is taken down to the porters by 2.30 each afternoon. The porters deliver the internal mail bundles to their respective colleges and external mail to the post office. External mail requiring priority action and is delivered direct to the post office *poste haste*.

The social organisation of the space's equipment, and the reflexive constitution of the space's distinct 'placeness' in that equipment's practical management and use, is further manifest in the orderliness of the refreshments area. As mundane pieces of equipment as they are, the coffee machine and kettle, are visibly ordered for public use. Notices make that public character and conditions of use *plainly visible*, informing users as to the price of beverages. At the side of the coffee machine stands a dish for monies.

Figure 4: Refreshments area

Again, a host of 'invisible' work carried out by support staff underpins the use of the refreshment area. Mundanely, in keeping the area stocked with beverages. Less so, in organising the routine maintenance of the facilities. Not all users of these facilities pay for beverages at the site. Different socially organised relationships are, then, implicated in the use of the space's equipment, relationships which are co-ordinated through different means. Members of staff pay a lump sum towards the provision of beverages and upkeep of facilities every few weeks. Payment is collected by support staff and its passing is ordered through the use of a hard-copy (paper) sheet. As and when any particular member pays, a box under his or her name is ticked off on the sheet. The use of the refreshments area – i.e. what goes on there socially – is as one might expect, quite diverse. In 'going for coffee' members meet one another; greetings, pleasantries, and jokes are exchanged. Conversations are struck – about work, personal matters, football matches, dates, and the rest. Frequently talk 'overflows' into the seating area; people take a seat, relax, continue their conversations out of the way of the refreshments facilities, giving others unhindered access to them. The seating area has its own orderly features adding to the 'placeness' of the space. Notably, a panel on the wall displays popular and also new book covers, books related to the

organisational business of the place (i.e. sociology).[1] Two tables display leaflets about sociological topics, courses, and other related material.[2]

Adjacent to the seating area stands a public workstation. As with the rest of the space's equipment, the workstation (a Macintosh LC 475) has its own ecological order providing for its public use. Next to the workstation, on the left, is a hard-copy document providing general information about the workstation: it's public availability (office hours); guidance on putting and leaving personal material on the machine; who to see for help and advice; the primary purpose of the machine (providing access to network services); and something of a user manual for dealing with common problems and tasks (such as doing 'forced quits', 'restarting' the machine, and using the internet browser). Beyond its programmed orderliness, the desktop of the workstation has its own endogenous organisation. Running down the right-hand side of the desktop are a series of shortcuts and folders. From top to bottom: 1) the local drive shortcut; 2) Folder no1. containing shortcuts to Unix and Telnet email facilities, and shortcuts to Netscape Telnet, Felix, and OPAC online library catalogues; 3) Folder no2. which is labelled 'public' and contains two items; 4) a shortcut to Microsoft Word; 5) a shortcut to Netscape Navigator; 6) a shortcut to Eudora Lite, which is not set up; 7) Folder no3. which is labelled 'stuff' and contains no items; 8) the wastebasket; 9) one Word document. Notably, Folder no1. is invariably open with its contents on plain view, and 'placed' in the centre of the desktop.

Figure 5: The Public Workstation: Macintosh LC 475

[1] Outside the mixing bay, on the walls of the corridor, posters for research positions, courses, careers information, etc., are displayed.

[2] One might also note, not unimportantly, that there is an aesthetic to the space, which consists not only of the *arrangement* of the space's equipment but also of artistic posters and plants.

As with the space's other equipment, the orderliness of the workstation, its ecological arrangement, is implicated in a host of largely 'invisible' work. This time, however, it is not support staffs' work but users work.[1] Although users occasionally employ the workstation to work on documents, by far its most prevalent use is in its intended purpose: accessing network services.[2] Specifically, in accessing email accounts, in browsing the internet, and in searching online library catalogues[3]. The use of email facilities relies on security, command codes, and social conventions tied to the space. Although network services are publicly available, naturally there is a *private aspect* to their use. Email accounts are personal accounts and access to any account is by personal password. Accounts are set up by the University's Information Systems Services and may be accessed from any public site on campus. Thus, the use of the space's email facilities trades on a prima facie *collaboration* with others responsible for *managing access* to certain pieces of the space's equipment. Minimally, that collaboration consists in establishing the identity of the potential user as a legitimate user of that equipment. That is, as someone who has a 'right' to use the equipment. The actual email facilities available are old (if not outdated) and their use trades on *a socially organised set of commands instructing* users: who to contact in case of trouble; how to access a menu of known troubles; for selecting particular messages to read; for deleting messages; forwarding messages; returning to message list, etc. Files cannot be attached to messages and the command for retrieving messages in the first place is not displayed but assumed to be known.

As noted above, the use of email, and other network services, relies on 'social conventions tied to the space'. These conventions are embodied in an orientation towards one another of persons using and wishing to use the workstation's equipment. There is an *observability* to the use of the workstation's equipment which consists of persons using and wishing to use it, *monitoring one another's actions* through a glance recognising the presence of another, a question, the specification of what's happening, what's being done now, how long it will take, what's going to be done next, and the rest. There is, then, an *interactional negotiation* between the space's members as to the *use of* the space's equipment. A course of negotiation which relies not only on members being able to 'work up' a sense of what's happening now and what's happening next but crucially, on

---

[1] And, although not observed in the course of the present study, maintenance staff where necessary, although importantly this is often carried by members of the space's population with some practical 'know how'.

[2] Working on documents is constrained by the purposeful public character of the workstation. There is what might be described as an almost constant daily flow of persons wishing to access network services. The opportunity to work on documents is constrained by that flow. Persons wishing to work on documents for any length of time may do so at other sites in the university, which are expressly set up for such purposes.

[3] Use of the internet browser and online library catalogues will not be described here. See eSCAPE Deliverable 4.1 (1999) *Chapter One: Searching as a Socially Organised Achievement in the Library* for a detailed account of the organisation of online catalogue use. Use of the internet browser shares a great many similarities with use of online library catalogues insofar as members are searching for materials. Unique to browser use, however, is an evolving list of bookmarks, most of which are related to the space's populations' organisational business. Thus, bookmarks are stored for research funding, sociological associations, sources of research materials, other institutions, and the rest.

seeing at-a-glance and *being able to see at-a-glance* what's happening in the space, what's being done with certain pieces of equipment, and who else is located there.[1] Use of the workstation's equipment relies, in other words, on *an observably furnished sense of co-presence* which underpins subsequent negotiation and turn-taking.

Space and place in CSCW

The purpose in this section of the report is to consider the endogenous social organisation of the mixing bay in light of previous work in the field. Prior to considering the implications of the mixing bay study we want, then, to return to previous CSCW studies and to do so in order to place the study findings in a relevant context for design. As noted in the introduction to this report, the material arrangement of the space's equipment constitutes an 'ecology of awareness' important to the development of CSCW systems (Hughes *et al.*, 1994: 157). The concept is intended to draw attention to the significance of spatial arrangements as 'essential ingredients' in the day-to-day *doing* of the space's work. Material arrangements afford an intersubjective awareness of the space's work, of what is going on, what is and can be done here, of what needs to be done, and the rest. As previously noted, by 'intersubjective' we refer to the embodied *social* use and reflexive organisation of physical space and its material arrangements or composite equipment. This is not all there is to the matter however, as the notion of intersubjectivity does not simply refer to the sociality of spatial arrangements but to the ways in which setting's are *made* social by some population of members in and as of *the practised use* of the space's equipment. In other words, the material equipment of the space is irremediably connected to the *coordination* of individual lines of action in such ways as to produce an organisationally distinct and publicly available space. The 'connection' consists of various *ensembles of equipmentally affiliated work-practices* engaged in by the space's members, practices which in their performance lend the space its public character. Thus, the 'publicness' of the space consists in various ensembles of equipmentally affiliated work-practices which provide for intersubjective awareness and the coordination of activities. In light of such observations Hughes *et al.* (1994: 157) suggest that

> computer systems have electronic effects that result from the work taking place. In the same way that the physical effects of work are central in supporting the cooperation taking place these electronic effects are essential to the development of computer support for cooperative work. Understanding and supporting these equivalencies is a major design challenge for CSCW. One approach to exploring these effects is to investigate the nature of space and how the space is used to support cooperative work. This understanding can be used to developed electronic equivalents which are applicable to work taking place within electronic rather than physical settings.

[1] See eSCAPE Deliverable 1.1 (1998) *Chapter Five: Developing a Framework for eSCAPEs* for related issues here.

What might this mean? What might investigation of the nature of space and how space is socially organised amount to for the design of cooperative systems? Seen from the practical point of view of the space's daily work one thing it might mean is that the accomplishment of intersubjectivity, of mutual awareness and co-ordination of individual lines of action in accomplishing the space's daily work, relies on *representing* the space's work as part of the work itself. The emphasis here is not so much on abstraction as on representing as an activity which is both essential to the daily accomplishment of the space's work and which can be done through various kinds of equipment comprising the work setting. Thus, and for example, the whiteboard is a piece of equipment for representing elements of the space's work in the course of that work. The point is that the whiteboard representations are not isolated and detached (abstract) but embedded in the work itself, reflecting the details of that work so that it is available to the space's population. In other words, what representations do, among other things, is make the work 'visible' so that it can be 'taken note of', 'reviewed', 'queried', and so on, by others implicated in the accomplishment of the space's work. Representations put the work on display so that others may be aware of it and take action accordingly.

The overriding practical concern in treating the issue of representation is *what* to represent about the work, *for whom*, and *how* to represent it. In other words, there are principles of selection involved. For example, in managing the whiteboard, support staff do not write 'just anything' down. Use of the whiteboard presupposes an organisational relevance and relies on a mundane *orderliness*. In details of that orderliness the work, its details, its progress, its state of affairs, is made available to others involved in the work. Thus, the whiteboard becomes a distinct piece of equipment employed to engender awareness and support the co-ordination of individual lines of action amongst relevant members of the space's population. Equipment for representing situationally relevant features of the work of the place is essential to co-operation then. At a grossly observable level representation trades on a mundane orderliness, such as that implicated in reading, which makes the site's work publicly available. More subtly, the issue of representing work is also bound up with the kind of work that representing itself consists of – the 'invisible' work of representing - and what the representation is to be used for. As Garfinkel (1967) argues in respect of record-keeping, this is not some activity detached from the work itself but is 'an element of institutionalised practices'. What do such practices consist of with respect to the doing of representation work?

By way of elucidation, one of the features of work observed in the mixing bay study in connection with awareness and the co-ordination of activities is the way in which information goes through *modal transformations*. Take the whiteboard, for example. The information displayed on the whiteboard is not static but changes over time. The accomplishment of changes consists of the accomplishment of local work-practices, practices which are tied to the space and

productive of its distinct 'placeness'. As such, the local work-practices in and through which changes are accomplished are 'institutionalised' practices, practices which in their performance are *productive of* this space as a mixing bay *in a sociology department*, for example. The nature of those practices may go some way to inform us as to what the doing of representation work consists of.

The accomplishment of changes to information on the whiteboard consists of the co-ordination of independent lines of action between staff responsible for managing the whiteboard. As noted earlier, coordination trades as a matter of daily *procedure* on the use of specific pieces of equipment – electronic and paper diaries in this case. It is not enough, however, to make the information in diaries available, that information must be *presented* in an *economic*, publicly available *format*, as provided by the whiteboard. Thus, one thing that can be said about the doing of representing work beyond the fact that it consists in the accomplishment of equipmentally affiliated institutionalised work-practices, is that although the particular things represented will vary from place to place according to the 'placeness' of the space, representing work nonetheless consists of the use of *standard procedures* in the accomplishment of 'invisible' work and *standard formats* such as the orderliness of the whiteboard in presenting information and making the space's work public. This is a generalisable feature of representing work that suggests potentials for the design of co-operative systems.

In addressing the use of standard formats in documentary based representations, Hughes *et al.* (1994) identify a number of features relating to formatting in connection with awareness and co-ordination in design. In the first instance, they point out that formatting is not only a ubiquitous property of such representations but more importantly, *a social property* in a number of senses.

> First, [formatting] imparts an element of standardisation so that the information it presents can be shared. Second, many formats are widely used as conventions within many activities and, thus, widely known. Third, a format is a set of procedures for creating and using representations … 'Knowing the format' is tantamount to knowing how to use it to create and use it as a means of representation. Knowing how to use a spreadsheet, for example, means knowing how its data is arrayed in a vertical and horizontal manner, knowing how to move between the rows and the columns, and so on. (Hughes *et al.*, 1994: 154)

Thus, one of the properties of formatting is the way in which it can *proceduralise* representation and, through this, represent the work to others for particular purposes. Notably, in such ways equipmentally affiliated formats work as *mechanisms of interaction*. As such, and furthermore, formatting is one of the ways of practically settling the issue of relevance; that is, answering the question what should be represented and how? The format of the whiteboard, for example, lays out for its users what is relevant, much as the category organisation of the post boxes does. What to represent and how is irremediably tied to *the kind of*

work activities formats are intended to represent (and that, naturally, depends upon the work of the space).[1]

Although formatting as a means to constructing representations is an answer to the practical problem of relevance within a domain, it is not an answer 'once and for all', but can be a source of 'troubles'. Formatting plays different roles as it were in the accomplishment of the space's work. In other words the same format is employed by different organisational personnel for different organisational purposes. Thus, support staff employ the whiteboard format to 'announce' relevant aspects of the space's work whereas staff and students employ the format to 'appraise themselves' of relevant aspects of the space's work. The problem here is one of articulating representations within the flow of work such that they are, for users, a natural and integrated part of the work rather than an 'unwelcome overhead'. The unwelcome nature of the matter lies in the value particular information has amongst different groups of the space's population and what they want the information for. Support staff, for example, need to be able to co-ordinate their work but the workings of that co-ordination do not and need not be available to others in the space; indeed they would be nothing more than an unwelcome irrelevance. The point here is in supporting the 'invisible' work implicated in doing representation; in providing support for accomplishing 'invisible' work but not cluttering the space with its equipment.

The intersubjective and public character of representations is dependent upon the use of agreed 'standards' and 'formats' of representation. Furthermore, and has hopefully been demonstrated by now, use of formatted equipment to represent information is essential to the production and co-ordination of the space's work. Shared formats thus provide us with a focus for analysis and the means of developing new mechanisms to support co-operative work. The reliance on shared formats of representation also highlights a tension in the design of co-operative systems. Traditional applications are essentially individual in nature and decisions about their properties lie within the remit of single users. However, co-operative applications are shared rather than individual in nature and alterations in the properties of applications have effects that go beyond the single user. Consequently, insofar as we can identify and design shared formats of representation cum mechanisms of interaction, how are we to manage the relationship between the individual user and the group in the doing of the space's work? How, in other words, are we represent and control the extent of *tailoring* within the space across a community of users? This is not a theoretical question but one which may be addressed through consideration of the equipmentally affiliated work-practices which constitute the space's daily work. To summarise matters first however: the design of electronic landscapes supporting co-operative work requires the development of support for doing representation work. This will require the design of shared procedures and formats of representation and support

[1] As noted in the introduction, similar activities may occur across different spaces and places, which suggests that although we must attend to the particular in developing support for cooperative work, the possibility of developing common mechanisms of interaction clearly exists.

for the 'invisible' work that underpins the use of representations. Support for controlling and managing tailoring of the space will also be required.

The mixing bay study: some implications for design

Talk of the mixing bay study has raised a number of issues for design. Not least, what does the study say about the nature of physical eSCAPEs? All too often in design circles 'virtual reality' is construed of in 'science fiction' terms. Without wanting to unnecessarily limit the scope of virtual environments, in many respects, 'fiction' seems to be the operative word here. When constructing virtual environments, the real-world 'constraints' of time and space may be dispensed with. (indeed, building such realistic physical constraints into a virtual environment is a non trivial task that is yet to be demonstrated successfully at any convincing scale.) Practical action, one may hypothesise, may be accomplished in radically new ways within a virtual world. Such is the vision and promise. Taking a rather less sanguine view however, for all the hype surrounding such notions, little evidence of 'radically new forms of interaction' that have any practical value exists.

Science fiction visions and the less scrupulous marketeers seem to assume, albeit in essentially vague ways, that for all intents and purposes, virtual reality exists: that it is already 'here'. With this assumption in mind, all that remains, then, is to figure out how to utilise it concretely. However, this assumption is worth examining in more details: utilise what precisely? Just what is there that already exists? What 'content'? Just what *practical actions* can be accomplished in virtual space? Beyond a few impressive *technical* demonstrations, and less impressive laboratory experiments, just what can we ***do*** in virtual space? Just what has been accomplished? Beyond allowing people to 'browse' around virtual structures and perhaps to talk to one another in the process of doing so (which amounts to little more than passive viewing however often 'interactive' categories of description are invoked), the possibilities are actually rather limited. Not a lot, that is and to be clear, in terms of just what people can do 'within' virtual spatial structures; in terms of just what activities they can engage in and accomplish. The truly interactive virtual world, and the interconnection of a host of different interactive virtual worlds, is (to use a phrase) light years away at this point in time. And by interactive we mean virtual worlds in and through which people can together accomplish in their collaborations a whole host of *specific work activities*. Simply put, virtual reality has *no point of connection* with the ordinary world of *cooperative work* activities as yet. How, then, may we 'go about' establishing such a connection?

The problem is two-fold. First, is the issue of the underlying technology. Generating the most trivial of virtual environments places a considerable load on

even today's impressive hardware: making a virtual environment that is suitable for meeting real-world specific work activities (rather than providing what are essentially 'cute' demonstrations of what the technology is inherently capable of) is a daunting task, and one that requires lateral solutions to overcome the limitations of today's hardware. Many of the technological issues, however, are at least well defined even if solutions to them all are not yet available (we give consideration to some of these, and propose solutions to them in the next chapters).

The second and more complex problem, then, seems in some respects at least to be one of the way in which we think about virtual reality. Rather than taking it that for all intents and purposes virtual reality exists (albeit, as we have already suggested, in essentially vague ways), we assume that for all intents and purposes it *does not* exist? Then the question we are forced to ask is "what would it take *to make it* exist? How could we *produce* it? What would it take to produce virtual worlds as *a practical achievement?* Even more radically, what would it take to make that achievement the practical achievement *of users?* And as user trials with the TIC demonstrator tell us, the achievement has to be that radical. For a host of reasons including the subtle but ever-changing nature of space and its material arrangements, there is no way we can make all virtual worlds for all users. Not practically usable ones anyway. Instead, and in light of the contingently evolving character of spaces and places, we must allow *users to construct, populate, and modify* the very fabric of virtual worlds.

Another fundamental fault in our way of thinking about virtual reality seems to be in the attitude we take towards the real-world. The real-world of spaces, places and material arrangements is conceived of under the auspices of 'constraint'; as something to be transcended and dispensed with. Why? By what necessity? Because science fiction writers thought that would be a good idea? How about we turn that idea on its head? How about we conceive of the real-world of spaces, places and material arrangements not as constraint but *affordance*? From the point of view of practical action, we experience spaces and places as *familiar material arrangements* that we understand as meaningful, intersubjective, mutually intelligible and (thus) irreducibly social 'structures' of practical action (consider bus stations, supermarkets, football stadiums, etc.; or again, witness users *natural* orientation to the DLC[1], the Year 1 CityScape and the TIC). In observable details of their material arrangements, familiar 'structures' of practical action *enable us* to 'get around' and 'get our activities done' (and museums and art galleries are no exception, no matter how provocative they may seek be at an experiential level). The familiar organisation of space – its 'structure' - is *essentially tied* to the accomplishment of activities that take place 'within' it then (Crabtree, 2000).*

---

1 eSCAPE Deliverable 4.0 (1999) Setting Up the Legible City, *Towards a Common Methodology*, 58-63, Lancaster University: Computing Department..

* Crabtree, A. (2000) Remarks on the Social Organisation of Space and Place, *Journal of Mundane Behaviour*, 1 (1), p. 25-44.

The familiar organisation of space and place consists 1) in the *visibility* of the particular space's material arrangements, of roads, lanes, signs, signals, and the rest, for example, and 2) in the *practised and competent use* of those arrangements, of driving in particular lines, following signs, obeying signals, etc. (ibid.). In stark contrast to being experienced as a 'constraint', the material fabric of space and place affords practical action in real-time then, in providing us with mutually intelligible 'structures' of practical action. So enabling users to construct virtual worlds might reasonably be expected to consist of providing them with the tools to ***make familiar 'structures' of practical action.*** And it might be so in that such 'structures' draw upon users natural and taken for granted understandings of space and place; natural understandings providing for action's production and accomplishment in and through the practised use of the material arrangements that comprise the space's familiar organisation. In other words, in treating the real-world as affordance, we might seek to develop tools enabling users to make virtual worlds out of *familiar real-world representations*. In doing so, we enable users to create meaningful visual layouts out of a host of familiar material arrangements, thus providing for the *at-a-glance intelligibility* of any particular space and for engagement in the activities that take place there. Notably, the ability to provide for an at-a-glance intelligibility of the space and its activities through the use of familiar 3D representations offers potential *added value* over the abstract 2D representations which currently comprise the 'interface'. The source of the 'added value' lies in the use of familiar representational *forms* as resources for *making sense* of the space and its activities.

The ***particularity*** of familiar 'structures' of practical action cannot be over-emphasised either. The kinds of representational forms comprising the material arrangements of familiar 'structures' varies according to the activities that take place 'within' the space. Thus, and for example, if you wanted to make a bus station then you wouldn't do it by constructing a football stadium (although naturally and importantly, you might use some similar components). The representations created and familiar structures constructed will depend, then, on the 'organisational business' of the space, on what goes on their as a matter of daily routine. As such, users must be able to construct virtual spaces out of representations *relevant to* the activities of the space. This much the Mixing Bay Study tells us. It also tells us that particular representations are used *for particular activities* and that the use of particular representations relies on a host of 'invisible' work concerned with the *practical management and maintenance* of particular representations. Supporting the construction of virtual worlds thus requires that we support not only the creation of relevant representations but also *'editing'* of the representations created. The Mixing Bay Study again informs us that not anyone can edit representations. *Access control* is an issue to be reckoned with then. An issue that is extended in considering the nature of personal places in public spaces.

Work-practice in the Mixing Bay raises several interconnected issues here. Firstly, there are *various kinds* of personal space. Some are *completely private*,

accessible by individual control only (email, for example). Secondly, other personal spaces, while preserving a confidentiality of personal information to some large extent, are nonetheless *subject to management by third parties* (pigeon holes, for example). Thirdly, and with particular respect to publicly organised private spaces (such as the pigeon holes), private spaces are *visibly embedded in group spaces*. Technically, these issues raise questions as to the concrete nature of access control and in doing so suggest a need to develop tools that not only provide for the creation of private spaces by individuals, and the specification of access control by individuals, but also, the need to develop an 'access model' supporting the 'monitoring', 'management', and 'grouping' of publicly organised private spaces by third parties. In light of these considerations, it would seem reasonable to suggest that developing support for the user-based production of virtual worlds will minimally consist of developing ***a generic toolkit*** supporting the construction of *particular places*. That toolkit will, again minimally, consist of 1) an object library enabling users to import and create relevant representations in constructing the space; 2) an editor allowing users to accomplish the 'invisible' work involved in maintaining particular representations; 3) an integrated access control model allowing users to manage access to particular representations and groups of representations comprising the space.

Chapter 3: The Design of Placeworld

Jeffrey Shaw
ZKM | Zentrum für Kunst und Medientechnologie Karslruhe

Placeworld has been conceived as an exemplary integrative demonstrator that assimilates much of the work done in years 1 and 2, and constitutes a new and mature embodiment of a physical eSCAPE. During years 1 and 2 a number of physical eSCAPE prototypes were built, some with an artistic starting point - *The Distributed Legible City*[1], *Nuzzle Afar*[2], *I0_dencies*[3] - and another with a completely practical intention - The Tourist Information Centre[4]. The Mixing Bay study report in the last chapter was used to ground a selected number of features from these earlier works and both these sources formed the basis for the conception and design specification of Placeworld.

In the broadest sense Placeworld sets out to incorporate features we have identified as important constituents of a physical eSCAPE:

- a place where diverse places meet
- a place where people meet both in and outside the various localities
- a place whose embodiment is scaleable
- a networked/connected place
- a place which people can modify and/or add information to

While the work done during the eSCAPE project has indicated various ways in which these objectives could be realised, we have chosen one particular design strategy that we feel offers an optimum environment in which to carry out both an innovative and integrative approach. This design incorporates the following components:

- ***a user interface*** that extends the Panoramic Navigator by adding a projection system to it, and a fully surrounding cylindrical projection screen
- ***the virtual environment of Placeworld*** that takes the artistic work *Place – a user's manual*[5] as its design frame of reference

---

1 Schiffler, Pettifer 1999: "The Distributed Legible City" in: *escape deliverable 5.1,* Lancaster.

2 Fujihata, Blunck, Vincent, 1999: "Nuzzle Afar" in *escape deliverable 5.1,* Lancaster.

3 Schwabe, 1999: "I0_dencies" in: *escape deliverable5.1*, Lancaster.

4 West, Hughes 1999: *escape deliverable 4.2*, Lancaster.

5 Shaw, West 1999: "Place a user's manual" in: *escape deliverable 5.1 Lancaster*.

- ***the contents of the Placeworld*** virtual environment that inherits and extends many of the physical eSCAPE concepts ands prototypes previously worked on in this project.

In the following sections we shall consider each of these in turn.

The user interface and presentation system

The Panoramic Navigator (PN) is a patented technology developed by Jeffrey Shaw and the ZKM whose initial function was to provide a panoramic method of allowing visitors to the ZKM to preview and explore the contents of the ZKM building. Using a 360-degree rotatable touch screen coupled to a video camera, this augmented reality approach enables visitors to the ZKM to look around the building (via the live camera image) and use the touchscreen to access additional multimedia information that is attached to specific locations in that building.

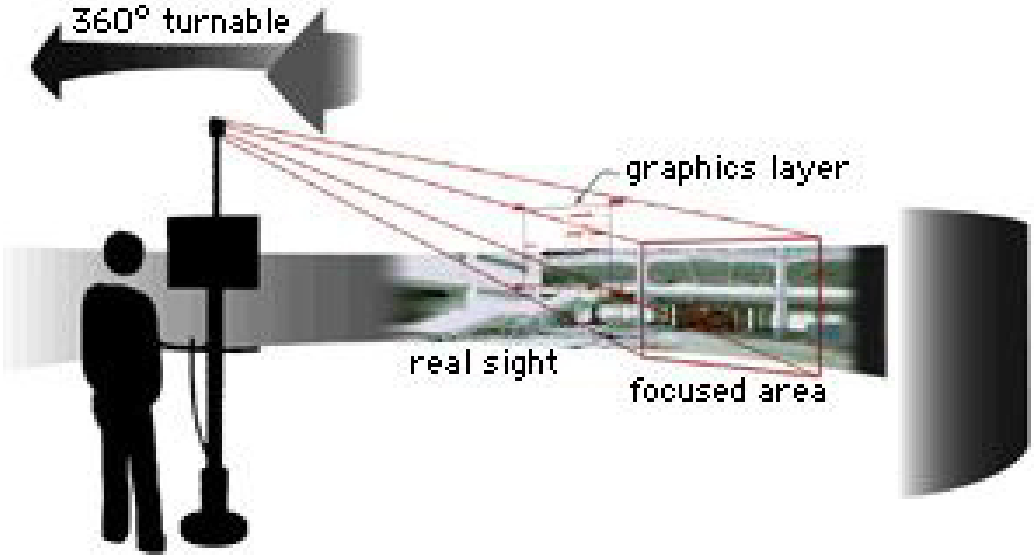


Figure 1: Diagram of the PN installation in the ZKM foyer, showing the live video capture of the surrounding environment which is then dynamically augmented by a graphics layer of information

At the year 1 i3 conference in Nyborg (DK), the PN was presented by eRENA as an interactive multimedia device to enable visitors to explore the contents of the eRENA deliverables. Here the real environment (a semi-circle of eRENA posters) was augmented by video clips that could be accessed on the touchscreen.

While these past implementations of the PN have used the real environment as its frame of reference for interactive information delivery, we realised that this technology could also be an ideal interface to explore and interact with wholly virtual environments, and that this would be eminently suitable for Placeworld. What was needed was a method of generating a representation of that virtual

environment around the PN, so that the touchscreen could then be used to interact with it.

The design solution for this was to attach a video projector to the back of the PN (behind the touchscreen) and to put a circular projection screen around the PN. In this way we have been able achieve a powerfully new presentation and interaction method which embodies the innovative concept of *augmented virtuality* (PN-AV). This PN-AV technology is used in Placeworld to embody two distinct but interrelated functions:

- to display the 360-degree immersive representation of virtual environment, wherein one also see the effects of user interaction with that environment.
- to give the user a touchable interface that allows him/her to explore and interact with that surrounding virtual environment by showing a parallel representation of that environment which is augmented by a dynamic set of user interaction tools.

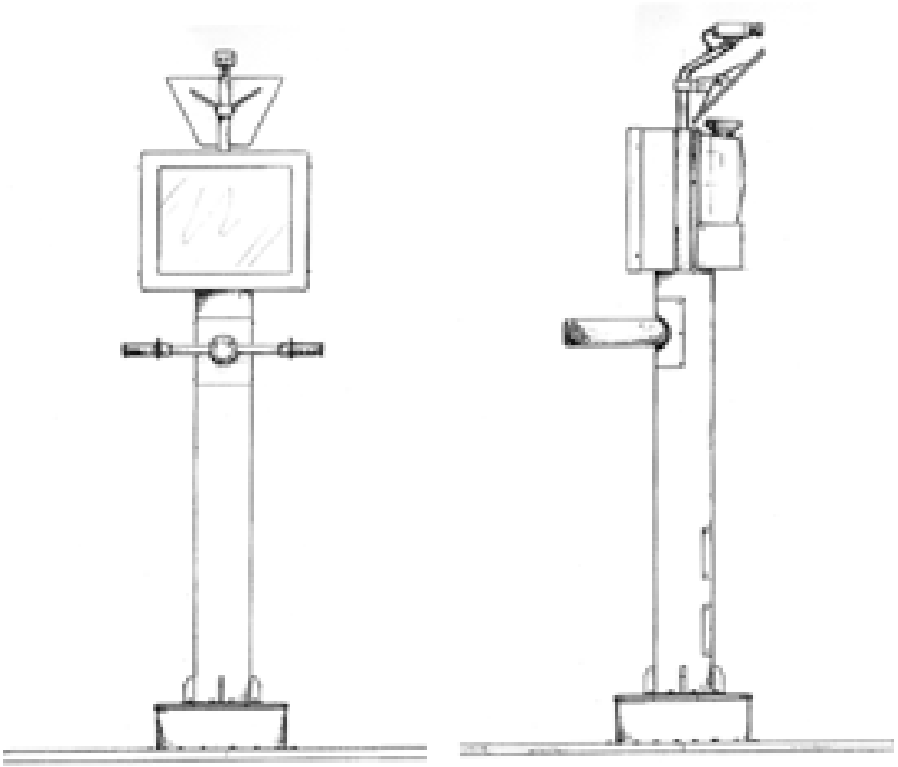


Figure 2: The new PN-AV showing the LCD touchscreen, the attached video projector (mounted vertically and reflecting off a surface mirror), and the multi-functional handlebar. The base incorporates a position encoder and electrical slip-rings.

The physical design of the PN-AV was dictated by certain considerations:

- the user interface rotatable column is ergonomically proportioned for adult use. There are of course various techniques (e.g. a moveable screen) that would allow such a device to be used by children and persons in wheelchairs, but this was not implemented at this stage.
- a 20" LCD touch screen was chosen for the advantages of its large size and high-resolution SXGA display.
- presentation quality SONY LCD projector was chosen because of its relatively compact size, brightness (1000 ANSI lumens), XGS projection resolution, and importantly its interchangeable ultra-wide angle lens.

Given the specific functionality that the PN-AV user interface needed in the Placeworld application, we arrived at the following specifications and design solutions:

1. the column itself rotates continuously in either direction, allowing the projected image to be moved around the 360 degree surrounding projection screen, and allowing user interaction at any physical position during this rotation
2. navigation of the virtual environment needs to have three degrees of freedom – rotation by rotating the column, variable speed movement forwards and backwards my means of a turntable handgrip, and lateral movement to the left or right by means of a tiltable handle attached to the column (which incorporates the handgrip)
3. the touchscreen should offer all the necessary auxiliary tools for user interaction with Placeworld by means of a dynamic icon and menu design strategy that seamlessly compliments the navigational and display characteristics of the PN-AV.

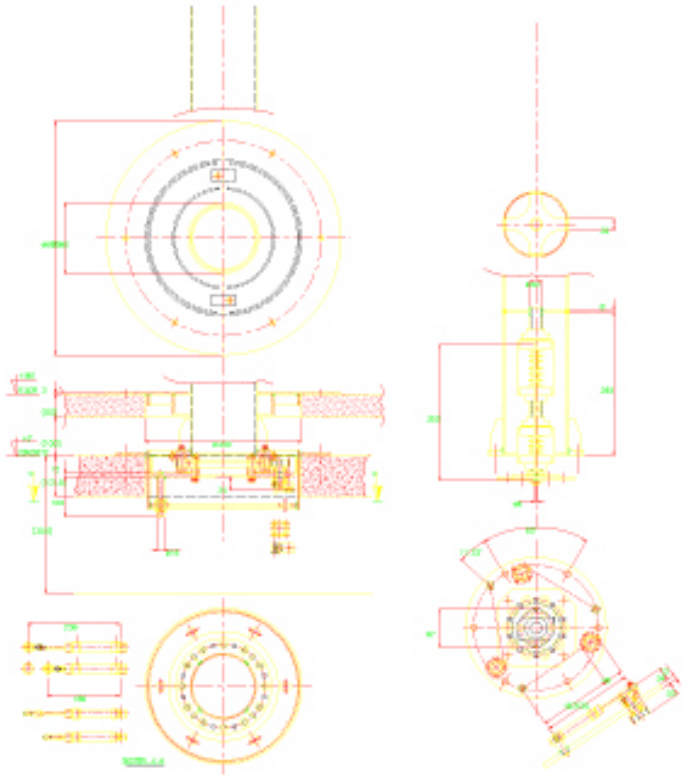


Figure 3: Detail of electro-mechanical construction of PN base unit.

The design of the PN-AV surrounding projection screen incorporates the following considerations:

- a PN-AV installation is by nature highly scaleable and one can imagine very large screen sizes (ca 10 - 20m diameter) that could address a very large audience. But in relation to the very limited space available at Jönköping we decided to specify what we felt was the smallest effective dimension: a screen of 3.5m diameter and 1.2m high.
- the projection screen material is suitable for both front and rear projection, which means that Placeworld, can be viewed from both inside and outside the projection cylinder. This is especially relevant for such a small diameter screen, which can only accommodate a limited number of viewers inside.
- the 1.2m high screen is hung so that there is a clearance of 1.5m down to the ground, which allows viewers to enter the cylinder by stepping under the screen. In this way the screen is a completely continuous surface, uninterrupted by an entry/exit opening.

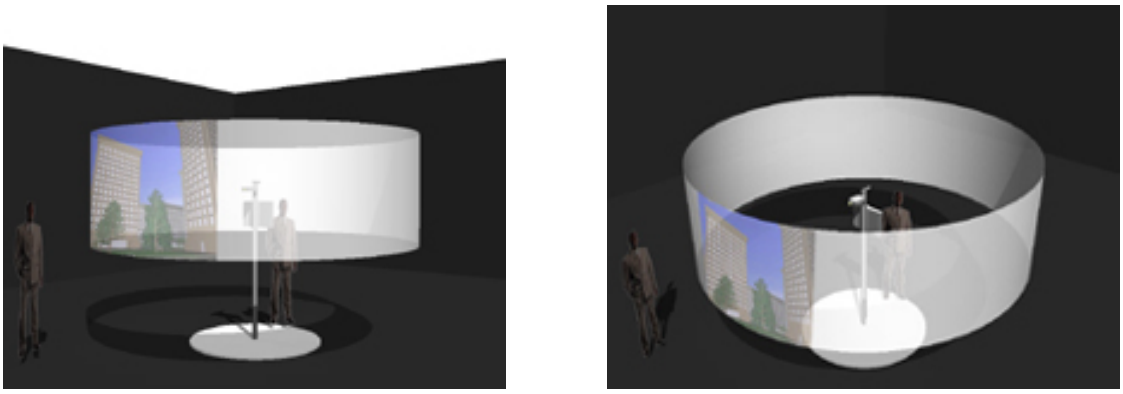


Figure 4: CG models of the complete PN-AV installation showing the PN column in the centre with its LCD touchscreen and attached video projector, surrounded by the 360 degree projection screen

The virtual environment of Placeworld

The design of Placeworld has taken as its point of departure many features of Jeffrey Shaw's artistic work PLACE – A USERS MANUAL (1995). The paradigm that is embodied in both implementations is that of a virtual navigable environment that is populated by an architecture of cylindrical form which can be entered and which in turn are portals into other virtual environments. Furthermore this virtual environment is characterised by pathways that provide a topography of navigable connectivities between these cylinders.

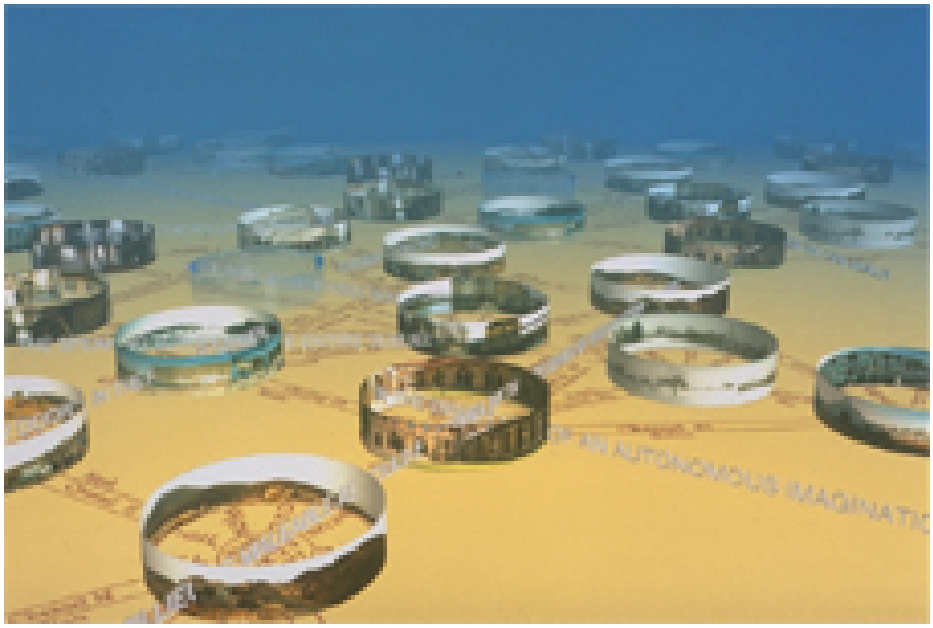


Figure 5: The navigable landscape of photographic cylinders in Jeffrey Shaw's art installation PLACE – A USERS MANUAL

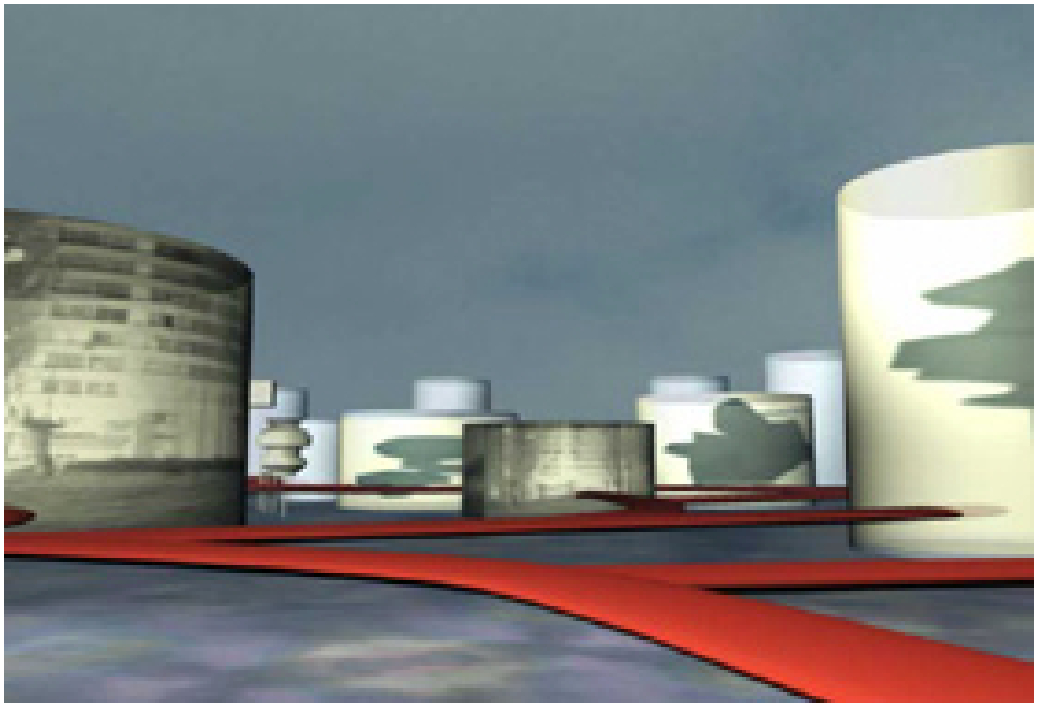


Figure 6: The first translation of the design into the MAVERIK application.

The decision to adopt this paradigm for Placeworld was driven by the following considerations:

- Placeworld, being a physical eSCAPE should be strongly referential to the ways in which the real world is structured and experienced, but because it is primarily an information space it should definitely not mimic reality. In other words, Placeworld should build on our everyday understandings of how real space is organised and navigated, but find innovative ways to extend these understandings into the virtual domain. Therefore we decided that the underlying real world parameters to be adopted are the physicalities of a ground plane over which one can move about in two axes, which embodies the visual sense of a landscape and wherein there is an urbanistic configuration of architectonic forms.

- Placeworld needs a coherent architectural design that is on the one hand systematic and easily understood; yet one, which is able efficiently, embodies a heterogeneous variety of disparate architectural forms and spaces. Therefore Placeworld, being a place where other places coexist, needs a special kind of architectural organisation .The cylindrical form is recognised to be a highly appropriate modular architecture for the exterior organisation of these divers places contained within Placeworld. To the extent that each individual place is a virtual environment in its own right, the cylinder offers a logical expression of the notion of boundary for such environments.

As boundary architecture, cylindrical space is quite different from more typical rectangular spaces. Whereas the former constitutes what is in effect a scaleable panoramic horizon, the latter is perceived as a performed enclosure. Therefore cylindrical spaces are especially appropriate within Placeworld because they allow all the idiosyncratic virtual environments to define their own unique enclosures within a neutral and scaleable circumference.

Despite the apparent exterior uniformity of these cylinders, attaching images to their surfaces offers an expressive way to signal their interior contents. In this manner (rather than by architectonic variation) they constitute a characteristic and very expressive visual syntax for Placeworld. This is especially important for purposes of orientation and accumulated familiarity for visitors within Placeworld. Furthermore, this cylindrical uniformity does not preclude the possibility to scale them variously in height and width, or to work with the relative transparency of their surface. These modalities allow a further articulation of spatial as well as temporal meanings to be expressed via this architecture.

The everyday urban landscape is as much characterised by its roadways as by its buildings. In Placeworld such pathways are also fundamental to its appearance and comprehensibility. A network of connectivities between the cylinders, such pathways are at the same time able to be dynamic informational structures that can for instance express the volume of user traffic by changing their width.

Having concluded that such an architectural landscape of path-connected cylinders would be an eminently appropriate design strategy for Placeworld, Andreas Kratky (ZKM) has developed a design specification that takes this earlier work and develops and extends it with new features that address the particular needs set by the objectives of this demonstrator. One important aspect is the fact an everyday urban landscape is characterised by the presence (or absence) of its population of citizens. Placeworld also socialises its idiosyncratic landscape with representations of its visitors and occupants, but in accord with the inherent properties of this information space, these avatars have unique design characteristics that enable them to effectively function as true citizens of Placeworld.

The content design of Placeworld

A deep going process of discussion between the involved partners focussed on various options for articulating the contents of Placeworld. In the following part of this deliverable we shall report on the salient aspects of these discussions which formed the background to the content structure of Placeworld that has been implemented and documented in the previous chapter.

Navigation

From the outset we decided that the navigation parameters for a visitor to Placeworld should be closely related to navigation modalities in the real world, but should be reduced to those degrees of freedom that are needed to intuitively and efficiently explore the specificity of the Placeworld environment. Studies of various navigable virtual environments in relevant artworks were used to guide the precision of these formulations.

The navigation features decided upon were:

1. the viewer's point of view would be straight ahead at adult eye level.
2. a moderately wide viewing angle would be offered (though interest was expressed in the notion the each user could interactively control his own angle of view, especially to enable ultra-wide outlooks as an orientation aid).
3. freedom of motion in the x and z axes by means of control of direct forwards and backwards movements (preferably with speed control) in the z axis and translation to the left or right in the x axis (here a fixed speed would be adequate).
4. the viewer can rotate his point of view continuously either to the left or right, and all movements are constantly relative to the viewers momentary point of view

While these navigation parameters were felt to be sufficient and appropriate for the Placeworld space exterior to its cylinders, we realised that these may not always be suitable for the disparate virtual environments that were being created within the cylinders. In those cases it should be possible to redefine the navigation modalities (of course within the constraints of the generic interface hardware design that we had adopted). And such parameter changes should be clearly signalled to the user on the touch-screen interface so that he/she could easily comprehend the altered conditions that might exist within the virtual environments contained by these cylinders. For users who would create their own environments with idiosyncratic navigational properties, this explanatory obligation needed to be understood and implemented.

Further navigational features:

- the complete circumference of the cylinders in Placeworld are permeable and could be entered at any position.

- the interior scale of the virtual environments embedded in the cylinders could be freely specified – for instance *The Distributed Legible City* would be many square kilometres, whereas the Manchester oil rig is only some cubic metres. As a consequence it was realised that if the method of exiting an environment would only be through its cylindrical circumference, this might cause difficulty and frustration for the visitor in the larger environments. Therefor a touch-screen operated immediate exit option would be desirable.

Mapping, signposting, zoning, persistence, hyperlinks

Discussed also was the relative desirability of offering the visitor (via the touchscreen) the option to obtain a bird's eye view of the whole Placeworld environment as an orientation and navigation aid, as was implemented for instance in *The Distributed Legible City*. But *The Distributed Legible City* also showed the limits of this facility – that it was useful as a method of taking one's current bearings, but that it could not be used as reference for controlling one's actual movement through the virtual environment because of the great dissimilarity in scale between the map and that environment.

A visitor in Placeworld would be expected to orient himself/herself to a largest extent by the visual identities of the cylinders themselves – i.e. by the individuated images that would be textured on the outer surfaces of these cylinders (a form of advertising their interior contents). But as in the real world, it seemed useful to also establish some method of signposting in Placeworld that could offer and guide visitors to the various locations. Such signposts could also be set up by persons who had built a place in Placeworld as pointers to their location. A problem could be the inherently limited graphical resolution of the overall representation, so particularly text signposts in Placeworld might have to be inconveniently large.

Another navigational aid that was discussed was the notion of zoning – i.e. clustering related environments in specific neighbourhoods. Such a zoned organisation of Placeworld would then become a component of the *Access Model* (see below). Here of course real world orthodoxy's such as the shopping mall, the residential area, downtown, the business quarter, the red light district, the University campus, etc. could be adopted and extended.

The issue of the relative permanence of the overall geography of Placeworld and its cylindrically bounded environments was felt to be an important issue if regular visitors to Placeworld are to be expected to readily gain familiarity with it. As in the real world, our homes tend to stay at the same addresses that they were built. On the other hand, the real world allows for the entropy of its structures over time, and caravan homes do move to new locations! So there are two issues here: one is the real need to establish a long term persistence of the organisation of Placeworld, but with the possibility of changes that could be warranted for

instance by reason that a certain place is never inhabited or visited, so it may as well shrink down and disappear (or be relegated to a 'dump' zone).

A point of debate was the notion of hyperlinking the individual cylindrically bounded environments so that a visitor could zap from one related place to another without having to negotiate the Placeworld landscape. And a more radical extension of this facility would be that a visitor could at will completely reorganise the distribution of cylinders in Placeworld to suit a specified personal need or desire (e.g. he only wants to visit the popular places, so these all become clustered around him and the others disappear). But going back to the premise of Placeworld as a physical eSCAPE that should be strongly anchored in real world physics and experiences, it was felt that such hyperlinks and hyper-rearrangements were not consistent with the objectives set for this demonstrator, and that they could be better and more appropriately explored in the other eSCAPE demonstrator, the 'abstract eSCAPE'.

Traces and other temporal markers

Masaki Fujihata's eSCAPE project *Nuzzle Afar* gave us an exemplary demonstration of the value and effectiveness of attaching traces to people's movement through a virtual environment. Colour coded for each visitor, these lines gave temporal evidence of people's presence and their paths of interest. And the added feature that a visitor in such an environment could 'hook on' to someone else's trace and then get a roller coaster ride along that convoluted line that brought him face to face with that other person was also very effectual (and enjoyable). But further discussion of this methodology revealed a basic weakness – as soon as there would be a large number of visitors in the virtual environment these lines would become an unintelligible tangle. Because we had decided to restrict movement to ground level (x and z) in Placeworld, such traces could then only be drawn in one plane – adding to the confusion if there were many visitors. On the other hand the restriction of these markings to one plane revealed the possibility of another very interesting design strategy. We realised that these traces could in fact be merged with the geographical and physical articulation of roadways in Placeworld. These roadways could be programmed to dynamically and temporally express the volume of their usage. The hierarchy could be say trail, path, road, highway, which would be defined by the density of traffic experienced along these routes, and would change according to the varying density. In this way a new feature of informative comprehensibility could be added to Placeworld - the nature of the pathways would themselves express a history of visitor interests, so one could for instance choose the well trodden paths, or venture along trails into relatively disinterested regions, or forge a new trail into previously unexplored territory.

Following this line of thought about the informative value of temporal signifiers in Placeworld, we discussed the notion that the cylinders themselves could also express the temporal condition and/or the popularity of their contents. Design strategies such as relative transparency, colour, and luminance could be

used for this purpose. The argument might be that their outer textures fade or peel like old wallpaper because of age or disinterest, or that newborn and well-frequented places are glowing with vitality.

Importing and constructing your own places in Placeworld

In principle Placeworld is a shared virtual environment that offers any of its visitors the opportunity to claim a piece of real estate and build whatever environment they wish within a cylinder whose outer surface they would appropriately texture to announce what's inside.

An ideal hardware and software architecture for Placeworld would allow visitors to import and/or construct models and images in any format, but such an undertaking would be beyond the scope of what could be achieved (or is needed) by this short term research project. So a limited range of import and construction tools should be realised so as to simply demonstrate this principal functionality of Placeworld. This should embody demonstrations of the importation of existing models in various formats, as well as a design strategy that allows visitors to access tools that allow them to build places from scratch. The latter option suggests the notion of 'do it yourself' stores situated in special cylinders in Placeworld, where one could pick up (or buy) various tools, components, readymade objects, textures and images with which to build and decorate one's own place. And extending this idea, visitors could also set up their own shops for this very purpose (e.g. for designer furniture or niche market pictures).

Access models

A certain level of law and order will be needed in Placeworld that would guarantee the technological and cultural integrity of its intended functionality and public usage. This access model should determine the scope of user access and action in Placeworld, protect its citizens interests, constitute the ecological equilibrium of its geography, maintain the aesthetic quality of its appearance, and offer optimum circumstances for social operations. Needless to say this is also an ideological issue, where we will inevitably build a Placeworld that embodies our own partisan values. So we should possibly aim to build a robust enough technological infrastructure for Placeworld that is flexible enough to be reconstituted using alternative access models as well.

Placing existing eSCAPE databases in Placeworld

As a final integrative demonstrator we felt that Placeworld offers an ideal opportunity to incorporate a lot of the work done during the eSCAPE project. There is also an aspect of efficiency in being able to re-deploy already existing databases, allowing us to then concentrate on Placeworld's overall architecture and functionality. Candidates for inclusion in Placeworld included the Distributed Legible City, a model of an oil rig created by The University of Manchester, a modified version of the Tourist Information Centre, Q-PIT, possibly some 360-

degree video technology developed at the ZKM (if the required OpenGL driver for Linux becomes available in time, which it hasn't), SICS's Blob space, The Mixing Bay, an online access space like *The Web Planetarium*, and another artist's work *Memory Theater VR* by Agnes Hegedues.

What will be aimed for in the selection of environments is an exemplary demonstration of the heterogeneous variety of places that can be coherently conjoined within the Placeworld architecture. Not so much a lush or exhaustive variety, but just enough to make Placeworld operational, interesting and a successful exposition of principles.

While there is an already existing familiarity within the eSCAPE project for most of the environments planned for Placeworld, this is not the case with one of them – *Memory Theater VR* – so it warrants further description. The Hungarian artist Agnes Hegedues created this artistic work especially for the ZKM's Mediamuseum. It is felt to be especially appropriate in Placeworld because it already and coincidentally embodies an architecture of cylinders in which diverse interior environments have been excellently articulated. It is a virtuoso VR work that includes complex 3D models and animated objects as well as photographic and cinematic textures, all of which constitute a coherent conceptual and aesthetic whole. Therefore it offers Placeworld another good example of what a carefully crafted environment can look like within its cylindrical constraints. Furthermore, Memory Theatre VR was originally an SGI Performer application that will be adapted to fit into Placeworld’s Linux/Maverik environment – demonstrating the ability of Placeworld's software architecture to incorporate applications made with other tools.

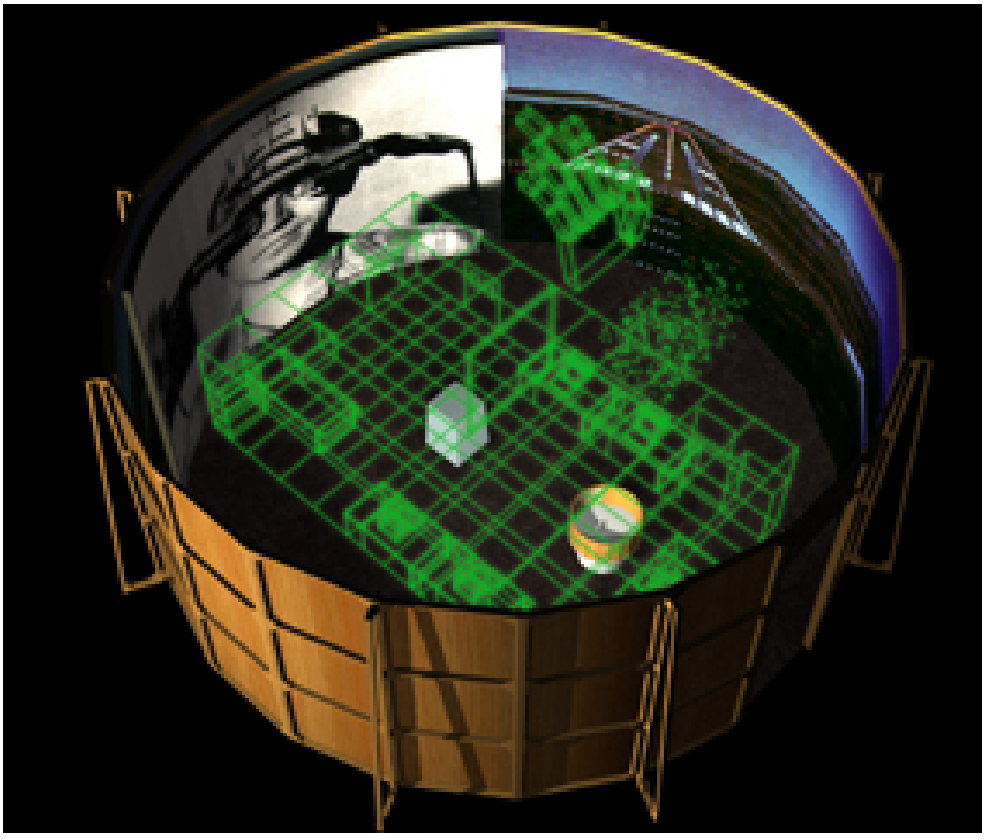


Figure 7: Memory Theater VR. Agnes Hegedues, 1997. Collection ZKM Mediamuseum

Functionality of the touchscreen

A major component of Placeworld is the design and functionality of the LCD touch-screen user interface. As indicated earlier, this device embodies the innovative notion of *augmented virtuality by* on the one hand showing exactly what is being projected externally as the representation of the Placeworld virtual environment, while at the same time offering additional informational, navigational, and functional features that are dynamically and spatially affixed to that representation.

There is a specific design challenge in this configuration. The touchscreen must offer a simple, generic and intuitive set of tools to enable any user to enter and interact with Placeworld, embodying all those functionality's that have been set as the modus operandi of Placeworld. These tools shall exist on the 2D surface of a relatively small (but high-resolution) screen, and at the same time be affiliated to the larger scale representation of a panoramic 3D space that is Placeworld itself. So the design convergence of 2D and 3D modalities of visualisation can be effectively researched here, and arrive at an appropriate and effective equivalence (complementarity).

Interpersonal contact and communication in Placeworld

From the outset it was felt to be necessary to offer representations of the visitors in Placeworld. We knew that corporeal realism was not a necessary prerequisite and only wasted computing power. *The Distributed Legible City* demonstrated the viability of simplified puppet-like figures, while evenly the extremely abstracted figures in *Nuzzle Afar* (just spheres) also worked well. What the various studies did reveal was the importance of giving these figures a physical indication of their orientation, especially when one wanted to have confirmation of face to face contact. Another property of these figures that was seen to be desirable was a method of personalisation - in *Nuzzle Afar* and *The Distributed Legible City* this was simply done by colour coding, but other more articulated methods of giving individuality to Placeworld's denizens could also be considered. Ultimately we realised that the characterisation of the visitors bodies in Placeworld was a design challenge where we had the freedom to invent forms that should be simply attractive, identifiable, contextually functional and graphically economical. And there should be an aesthetic consistency between these avatars and the nature of the overall Placeworld environment.

The possibility of interpersonal communication between visitors to Placeworld was a fundamental requirement from the outset. *The Distributed Legible City, Nuzzle Afar* and other studies all affirmed the effectiveness of direct voice communication. But in contrast to the acceptable reduction of the visual representation of persons to 'low fidelity' forms, one expects (demands) high fidelity on the aural level because a reduction of audio quality leads to difficulty in understanding and user frustration. In both *The Distributed Legible City* and *Nuzzle Afar* technologies were developed to allow adequate fidelity voice communication between a small number of persons. But this consortium chose not to develop the far more complex technology needed for qualitative networked audio communication between larger numbers of persons. Instead, to satisfy this need, we decided that in Placeworld people could communicate by means of well-designed text balloons and simple keyboard input (a virtual keyboard on the touch-screen).

One of the most important design achievements of *The Distributed Legible City* was the method developed to allow visitors in this large virtual environment to locate each other despite the fact that they may be hidden behind architectural features. This was done by means of birds in the sky that were attached to each visitor, as a pointer and guide to those people's locations[1]. This could also be a desirable feature of Placeworld, in a similar bird-like or other design form. Another interesting aspect of this concept is that the groupings of such birds in the sky would indicate, at a distance, areas of social congregation below, i.e. they would reflect the changing distribution of social activity going on at ground level. This would be especially useful if one wanted to find a quiet spot, or alternatively head to where the action was. Like the avatar bodies themselves, these sky born

[1] Schiffler, Pettifer 1999.

'alter egos' could also be individuated - for instance to indicate gender, age, personality or personal interests. And lastly, we considered the notion that these creatures would always be present in Placeworld - a permanent fauna - that would simply fall asleep (in perhaps a special 'aviary' cylinder) when their owners were not present, and await their return. This could also be a way to check to see if someone you were looking for was there or not.

The scalability of Placeworld

The software technologies developed for Placeworld are in the first place intended for distributed general use with generic Internet connected PC systems. At the same time we felt it would be useful to create a custom hardware configuration that responded to the particular needs of a larger public exhibition situation - in this case the i3 conference. This would demonstrate the scalability of the application, and its ability to migrate between generic and custom presentation formats. Furthermore the Panoramic Navigator with it Augmented Virtuality features should be understood as an appropriate research extension of the Placeworld concept, where the uniqueness of its contents is also be embodied by the design of a unique interaction and visualisation technology. In this way we could respond to the growing importance of distributed multimedia facilities in the broader urban landscape whose design should be tailored to the specific requirements of location based situations (urban furniture, fairs, museums, entertainment centres, etc.).

Chapter 4: Structure and User Interface of Placeworld

Andreas Kratky
ZKM | Zentrum für Kunst und Medientechnologie Karslruhe

The previous chapter provided a general description of the design of Placeworld and the ideas underpinning it. These ideas were developed in line with the results to emerge from the mixing bay study. In this chapter we wish to focus on the development of the user interface for Placeworld and the concepts underpinning the metaphors and interfaces realised within the demonstrator.

As we saw in the last chapter Placeworld is an interactive virtual environment for multiple users. It consists of a surrounding space that is populated by different cylinders (places) connected by paths. Each cylinder establishes an individual space and individual interaction strategies. Additional to the initial cylinder population of Placeworld, visitors can create new cylinders and thus construct and modify the system.

Architecture of Placeworld

Placeworld uses metaphors taken from the real world to allow for easy navigation and orientation in the environment. They make Placeworld legible for visitors because they refer to common experiences and at the same time they avoid any direct imitation of the real world. These metaphors and the interaction strategies implemented in Placeworld will be discussed in the following text.

The surrounding space is the environment that combines all elements of Placeworld. It contains the different cylinders, the connecting walkways and also artefacts that function like signposts and hyperlinks to other places or areas.

The surrounding space is characterised by a ground plane and the sky above the whole environment. These references to the real world allow for orientation and establish a sense of scale and directional movement. The ground plane is more a vague indicator than a real floor, thus all elements are slightly ‘floating’ above ground level. It allows for horizontal navigation and evokes a sense of gravity.

To circulate in the Placeworld visitors can either use paths or they can find their own way through new undeveloped areas. A certain number of paths are already provided in the initial setting. By navigating through new areas people leave trails, which can be followed by other visitors. When a trail is used several

times it is transformed into a path. Thus the different areas of Placeworld are structured by a dynamic system of paths representing the traffic flow.

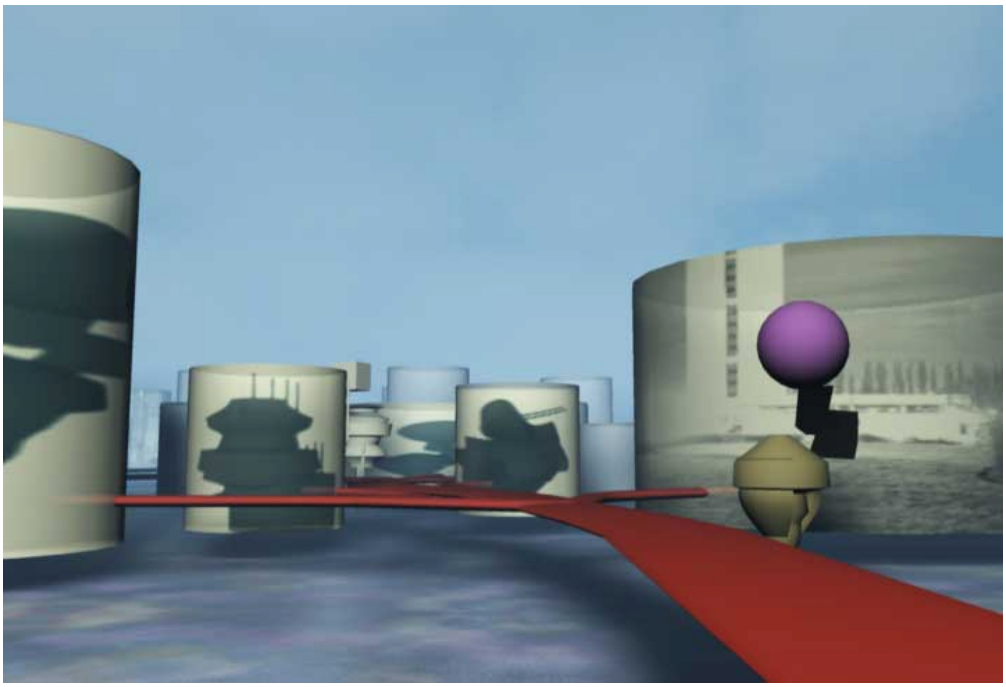


Figure 1: Cylinders connected by paths

The surrounding space contains all cylinders. It is organised in certain districts. Like in real cities these districts define general criteria concerning the use of certain areas. Possible zones are:

- residential areas (containing individual places created by visitors),
- service areas (such as the arrival court, the area where all visitors of Placeworld start their visit, the information cylinder giving explanations on Placeworld etc.),
- industrial areas (containing generator cylinders, cylinders containing external applications etc.)
- meeting areas (space where a big number of people could meet, containing 360 video places etc.)

Cylinders

The cylinders are portals to other environments that are independent from the surrounding space. They have clearly defined boundaries to the surrounding space but the inside of the cylinders can have a completely different size and can use different interaction strategies. The universal cylinder architecture allows us to accommodate heterogeneous objects and spaces. The hull of the cylinders is semi-

transparent and shows a texture that is significant for the contained objects allowing to identify the different cylinders and providing orientation in the environment.

Four different kinds of cylinders exist in Placeworld:

- individual cylinders (created and equipped by visitors)
- generator cylinders (cylinders that produce 'raw' material for the equipment of individual cylinders or for artefacts etc.)
- artwork cylinders (cylinders containing artwork like the "memory theatre", the "web planetarium", the QPIT etc.)
- YOU cylinder (cylinder providing tools to customise avatars)

The cylinders can be entered anywhere from the surrounding space by just navigating through the hull into a cylinder. As the inside may be very big there is an exit possibility provided in the user interface, which allows leaving the cylinders at any time and location.

Initially all cylinders have a standard size. According to the number of other people visiting the cylinders they grow as long as there is enough space around them. If there are other cylinders too close the growth is stopped. If a cylinder is not visited over a certain period it shrinks down until it vanishes. In this case the space can be taken over from by cylinders.

Individual Cylinders

Every visitor can create new cylinders in Placeworld. A newly created cylinder is initially empty and can be furnished by placing artefacts inside the cylinder or by importing external objects or applications. It is possible to import different items created in other applications outside the Placeworld and run them inside of a cylinder.

The material that is available in Placeworld to create artefacts is produced in generator cylinders and can be taken from there to be brought into individual cylinders. Different kinds of raw material can be combined to create an individually designed place. The transport of material is done by means of the POCKET, which is a cache where items can temporarily be stored to be taken to other parts of the Placeworld.

The outside of an individual cylinder shows a silhouette of the objects placed inside the cylinder. The silhouette changes according to the perspective of the viewer (textured turntable).

The individual cylinders are owned by their creators and can only be modified by them except when modification is explicitly allowed to other visitors. This is controlled by the access model.

Generator Cylinders

The generator cylinders are “factories” for ‘raw’ material that can be used for constructions in the Placeworld. Each generator cylinder produces a specific kind of material. In order to achieve a big variety of constructions the objects produced by those generators have a modular design and can infinitely be recombined. The best way to achieve this variability and at the same time reduce the complexity of the tools involved into the creation process appears to be an evolutionary approach. Based on a general description of the material type it is possible to randomly change a set of parameters defining the actual shape of the produced objects. Anytime when a visitor enters a generator-cylinder a first object is generated. If not satisfied with this object the visitor can make new variations until he finds a variation that corresponds to his ideas. This variation can be put into the POCKET and e.g. brought to the individual cylinder of the visitor and placed inside.

With a very simple set of user interface elements it is possible to create an unlimited amount of different objects. These elements comprise a possibility to initialise the creation of a new object, a possibility to mutate this object and a possibility to access the POCKET. The user does not have to deal with complex editing tools and does not have to pre-imagine the object he is about to create. The evolutionary approach is a source of inspiration, which confronts the visitor with an abundant flow of new objects.

As examples for possible generators three kinds of generator cylinders are available in the Placeworld:

- Object generators (creating geometric objects of different shapes for the construction of 3D artefacts as equipment for the individual cylinders)
- Text generators (creating random words; the words created by this generator type are all based on the structure scheme of English language. This scheme provides a specific structure of how words can be composed, which syllables can be combined and which characters make up an acceptable syllable. Thus all words are pronounceable and imply that they denote an existing entity. As they are closely linked to existing English words the range of words produced by the generator varies between existing meaningful words and completely non-interpreted words. These words form a basis for further editing and the composition of phrases. The text can be attached to artefacts or billboards.)
- Image generators (creating random images that can be used as textures)

The texture of the cylinders hull represents the produced material type.

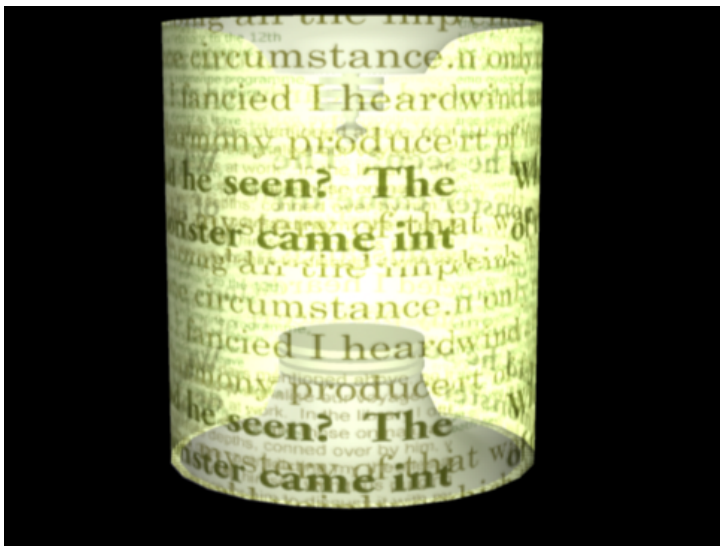


Figure 2: The outside of a text generator cylinder…

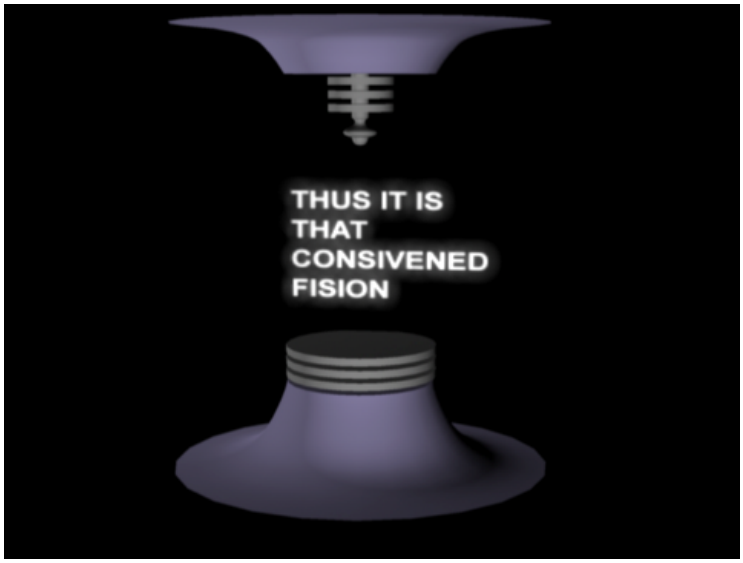


Figure 3: The inside of a text generator cylinder with generated text.

Artwork Cylinders

The artwork cylinders contain external applications imported into Placeworld.

Examples for the artwork are the “memory theatre”, the“oilrig”, the “web-planetarium”, the “legible city”, the “blob space” etc.

As an interface to other applications it is possible to create a new (individual) cylinder and import external objects into the Placeworld and run them inside a cylinder. Cylinders that establish a new, independent environment within Placeworld are artwork cylinders.

"YOU" Cylinder

All visitors of the Placeworld are represented by avatars. These avatars are only visible to other visitors. If a visitor wants to see his own representation and modify it he has to enter the "YOU" cylinder, which functions like a mirror. The "YOU" cylinder provides also the tools to make the following modifications:

- changing the colour of the avatar
- changing the size of the avatar
- applying procedural filters to change the avatars shape
- assigning textures to the avatars

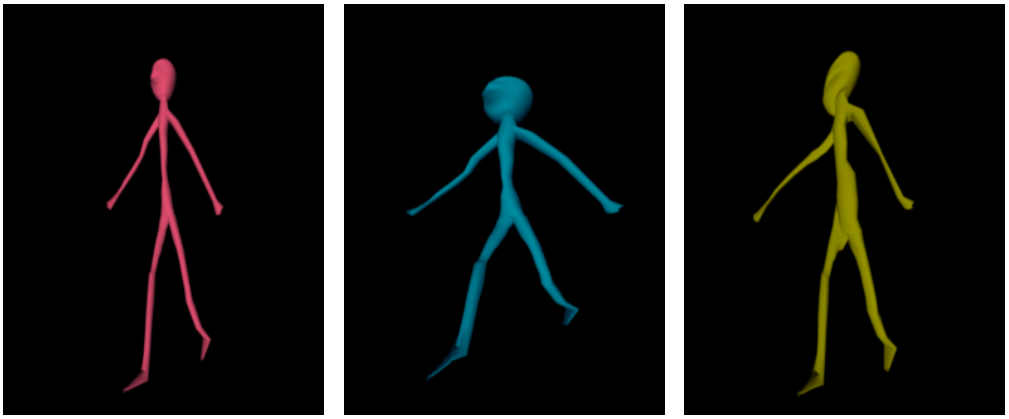


Figure 4: Modified avatars

Paths

In Placeworld the traffic of visitors and material normally uses paths. Although the visitors are not confined to the use of existing paths they often will do so. As a reference to real world ways and streets the movement on existing paths is faster than exploring new areas. The paths have a kind of magnetism that makes it easy to stay on the path once decided to follow it. Getting off is an explicit action and requires a lateral movement.

New paths are created when several visitors use the same way. By exploring new areas a visitor leaves a trail. The first trail that is left by one person is very small and nearly invisible. As soon as a second person uses the same way it is intensified. When several people used the same way their trails are consolidated to paths after a while. The more people use the same path the broader it becomes. The magnetism increases with the size of the path.

When paths are not used for a certain time they become smaller and smaller and finally vanish. This behaviour is similar to the real world development of ways. A single person leaves footprints and only when several people choose the same way an actual way results, i.e. when the destination is interesting. The network of ways is a collectively generated structure, which is a means of

communication. A person “writes” the information about a potentially interesting place on the ground and following people will read it. Existing traces or ways always will influence the creation of new ways. When a certain destination looses interest, people will go there less often and after a while the path-system that leads to these locations will disappear and with them the location.

Like the cylinders, paths can only cover the available space. If no more space is available they cannot grow and traffic-jam results. The constant flow of objects that are being brought to the individual cylinders and the people exploring the Placeworld creates a need for traffic structures.

The traffic on the bigger paths is automatically organised in lanes. The degree of traffic guidance and attainable speed increase with the size of the path. While trails do not provide any guidance for the user, big paths function like highways.

Avatars

In order to allow for interaction and communication between the different visitors of Placeworld every visitor is represented by an avatar and thus visible to other visitors. There was an intense discussion about the design of the avatars. They have to fulfil certain requirements specific to Placeworld. As the interior of cylinders can employ completely different navigation strategies than the surrounding space, the avatars have to be flexible enough to adapt to these requirements.

One example that establishes a completely different context than the physical environment of Placeworld is the abstract QPIT system. A special version of the QPIT system is present in Placeworld and lives in one of the cylinders. By entering the QPIT environment the avatar has to adapt to the abstract structure of visualisation and navigation. Outside the cylinders it is necessary to have at least an indication for two directions: the sense of up and down and the direction of the movement. In the QPIT system only the direction of movement has to be indicated as movement is free in all three axes and is not constrained to the ground. This is done by a short transformation of the avatars shape that takes place when the avatar crosses the hull of the cylinder.

The first approach to the avatar design was the idea to have abstract geometric shapes that can be transformed in many different ways. Avatars imitating the human body seemed to be too literal and not flexible enough to meet the different navigation and representation requirements. During the discussion we found that the abstract avatars solved the problem of transformation but they did not deliver the same impression of a populated space as more human-like avatars did. Also we felt that the difference between the normal furniture of the Placeworld (e.g. placed artefacts) and the abstract avatars was not perceptible enough. Thus it was the aim to combine the two characteristics to a humanoid but still abstract avatar and nevertheless still avoiding a direct imitation of the human body. The shape that was found at the end is a kind of stickman that has indicators for all qualities of the human body but still is an abstract representation.

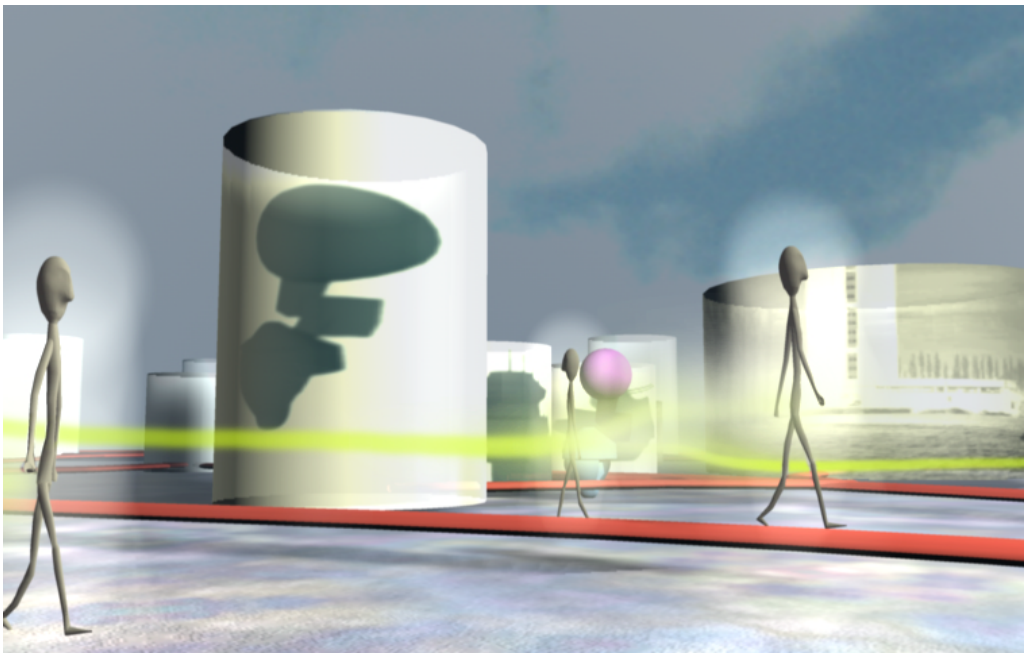


Figure 5: Avatars in the Placeworld

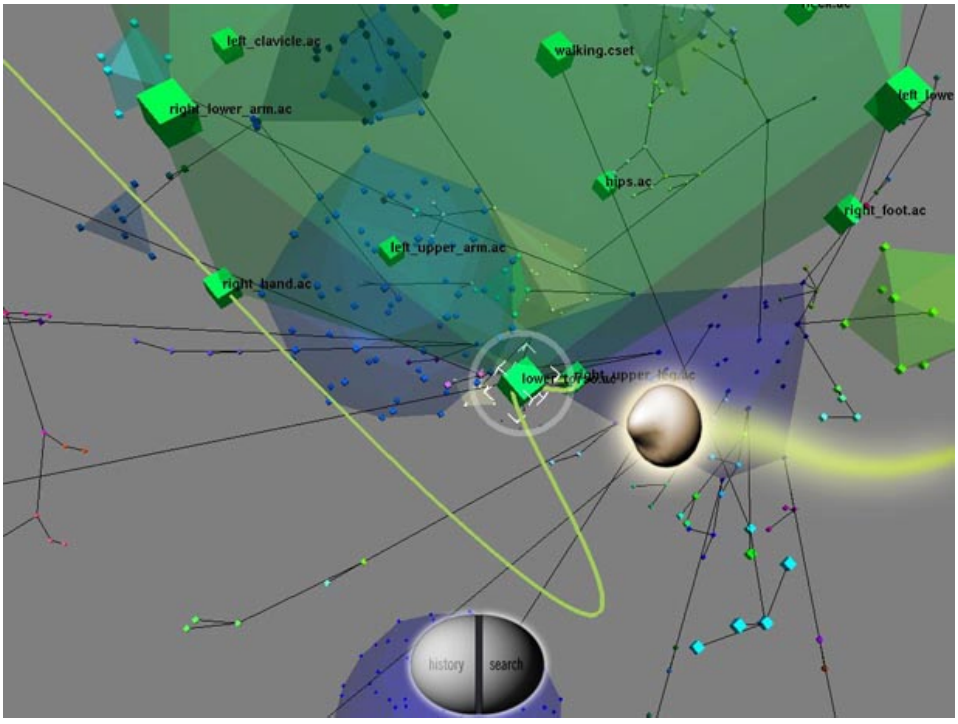


Figure 6: Transformed avatar in the QPIT

By crossing from the physical environment to an abstract environment the body of the avatars is reduced to the head. As the reference to a real person (arms

and legs etc.) is not necessary in the abstract environments the body fades away and only the head is travelling.

In order to distinguish the avatars as "living" entities from the rest of the environment they have a glow around them. This identifies them also as an element to interact with (see paragraph 3).

Navigating and Interacting in the Placeworld

The input device for the navigation in Placeworld is the Panoramic Navigator. The Panoramic Navigator allows three degrees of freedom for rotation, movement backwards and forwards and for lateral movement. It is equipped with a touch screen that provides all elements of the user interface required to navigate and interact with Placeworld.

The touch screen displays various interface elements like buttons and menus. Basically there are two menus, one called “history” and a second called “action”. They are accessible by an oval at the bottom of the screen, which is divided into two sections. This oval is constantly visible and stays the same in the different cylinders as well as in the surrounding space.

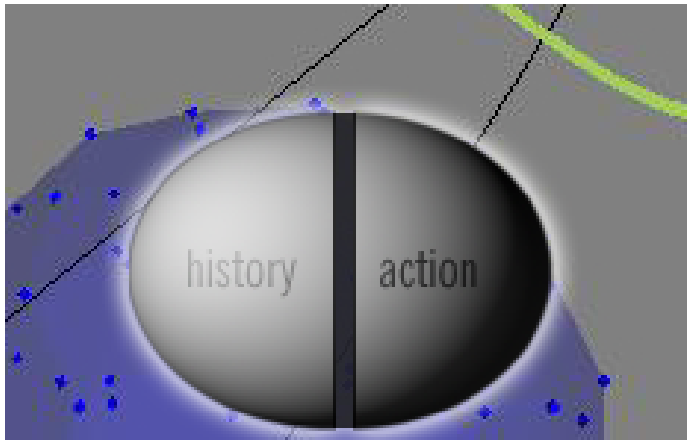


Figure 7: Main menu

The “history” menu provides the possibility to go back and forward to previous locations and, when the user is inside a cylinder, to exit the cylinder.

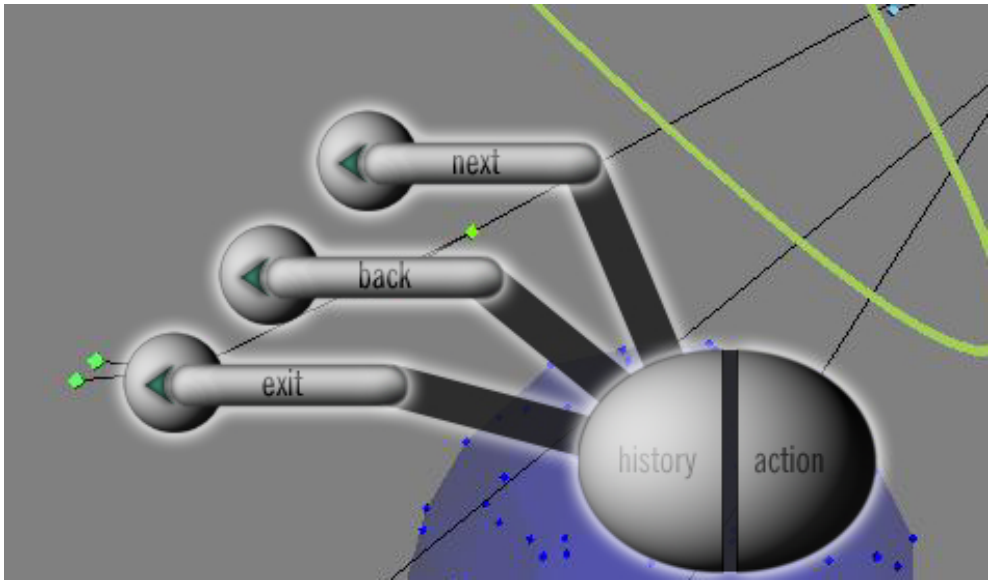


Figure 8: History menu

The "action" menu provides different items according to the cylinder the visitor is in. In the surrounding space there are only three menu items, which allow to create a new cylinder (this button is greyed out, when the zoning does not allow the creation), to access the POCKET and to invoke a short explanation of how to handle the controls in Placeworld. Commands that require more detailed specifications will open a dialogbox that allows the user to enter the corresponding settings.

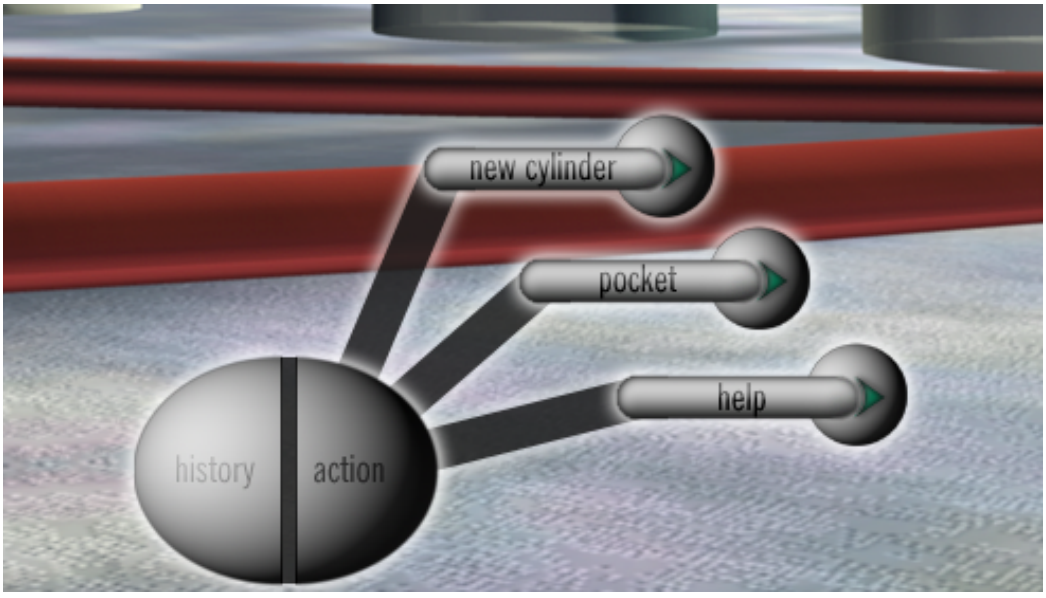


Figure 9: "Action" menu in the surrounding space

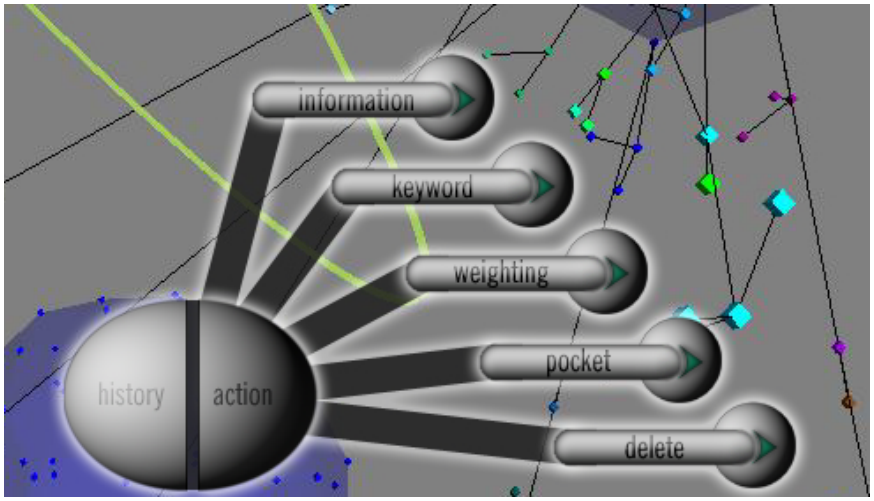


Figure 10: "Action" menu in the QPIT cylinder

All elements in Placeworld that can be touched to trigger actions have a luminous glow, which identifies them as 'hot' and at the same time separates them from the background. Not only the interface elements have this glow but also paths, communication links and avatars; as it is also possible to interact with elements located in the scene. Touching an avatar installs a communication with the person represented by this avatar. Touching a path opens a list of destinations where the path leads to and allows the user to use the paths to reach one of the destinations. Touching a communication link allows the user to join the ongoing communication.

The graphic design of the interface elements makes a reference to the material of the PANORAMIC NAVIGATOR and mimics the haptic quality of its handles. The size of the widgets is suitable to be precisely used with the fingertips. As the touch screen is fairly limited in its interaction possibilities compared to a normal mouse, all controls are designed in a way to execute all actions with a single tap. Sliders are equipped with two buttons that allow the user to increase or decrease values by simple taps.

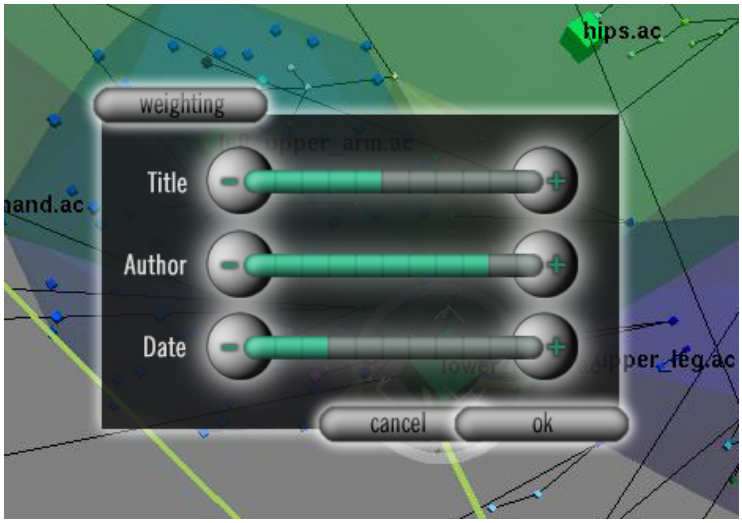


Figure 11: A dialog box containing sliders to specify the weighting of search criteria in the QPIT

A virtual keyboard that is displayed on the touch screen provides the possibility to type names for objects, to edit text and to write messages for other visitors. This kind of keyboard is not suitable for large amounts of text composition but it meets the requirements for text editing existing within Placeworld. E.g. it is possible to enter a keyword for a query in the QPIT environment.

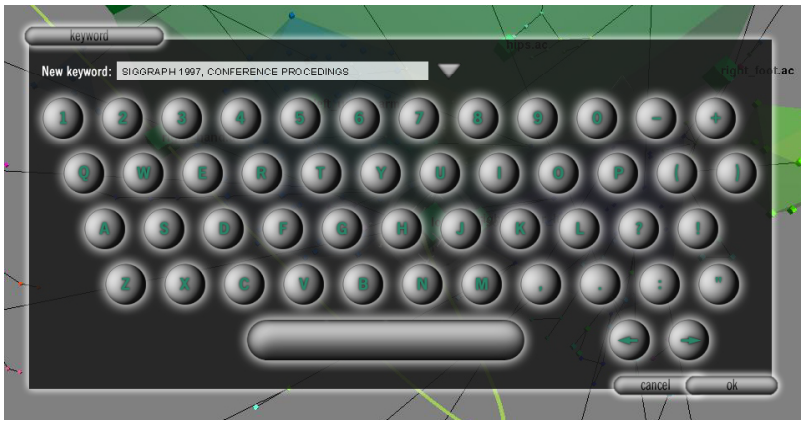


Figure 12: The virtual keyboard for all kinds of text editing in Placeworld

Independent from the path system visitors can also follow hyperlinks to specific places. Hyperlinks are set up by placing an artefact somewhere in the surrounding space that links to a certain location. The POCKET allows the user to add links to the temporary memory and attach them to artefacts.

Touching one of these artefacts brings up a dialogbox telling the user where this link leads to. If the user chooses to follow this link he will directly be brought there via a sky-link. Links do not use the existing paths but make “short distance flights” to the desired location.

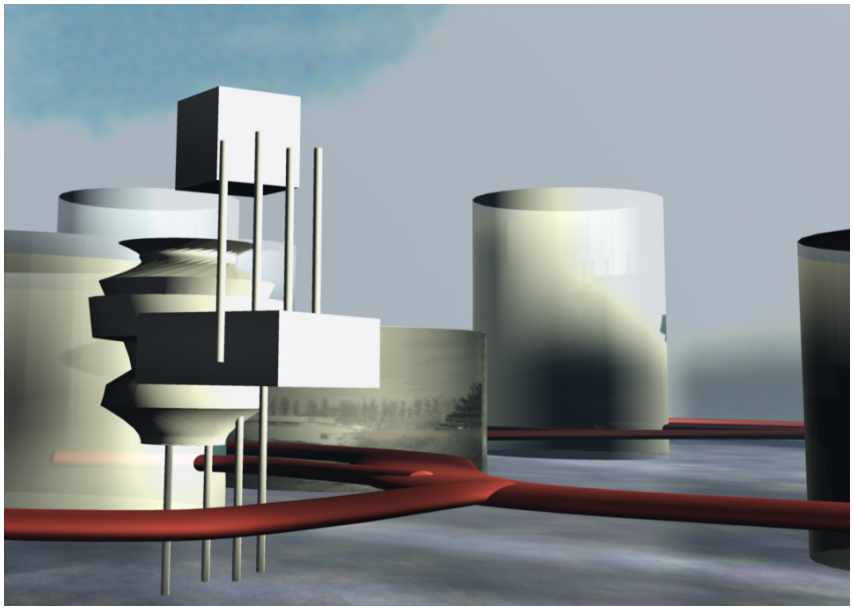


Figure 13: Artefact linking to a location in Placeworld

Communicating in the Placeworld

The paths in Placeworld function as a communication media as they embody information about preferences concerning certain areas and places of interest.

A second media is the direct communication between two or more visitors. It is possible to exchange written information by means of a chat-room.

Touching an avatar installs a communication link between the two visitors. This link is represented by a visible line that connects the two avatars. This line is visible to every visitor of Placeworld. Touching a communication line allows the visitor to join the communication. An unlimited number of people can participate in a communication. The chat-room metaphor and the exchange of written information makes a communication accessible to a great amount of people. The fact that the entire communication can be retraced in its written form corresponds to the materialisation of communication processes as visible lines.

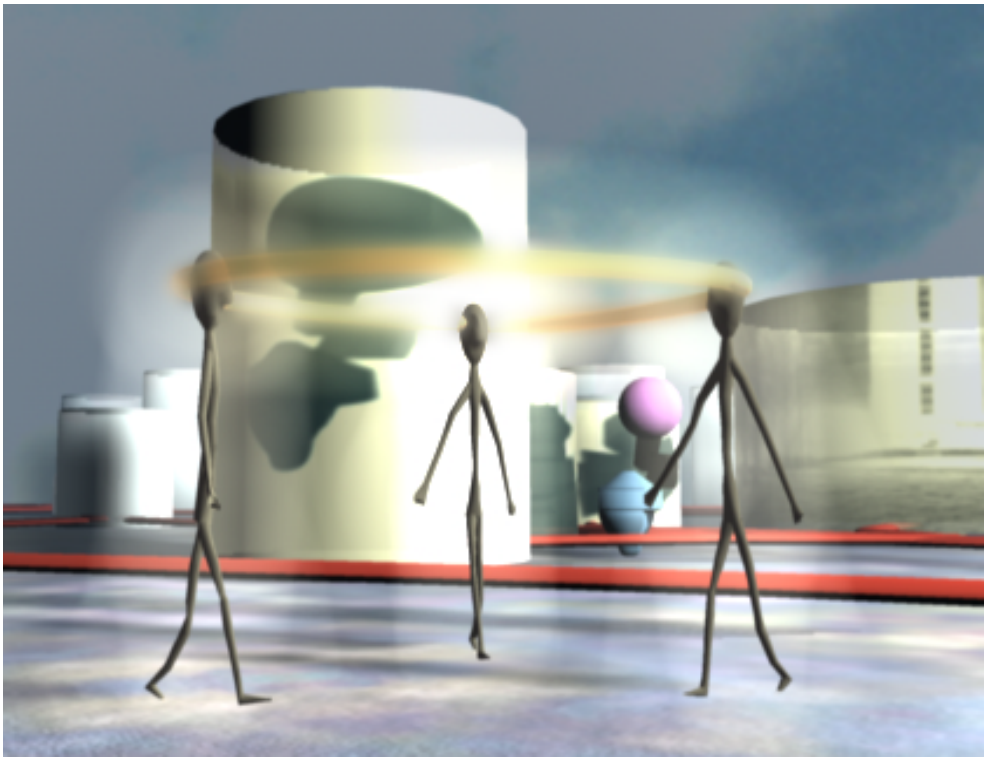


Figure 14: Three visitors exchanging their ideas in a communication. The written messages are visible on the respective touch screens

People can locate each other in the environment by following the trails they leave by navigating through Placeworld. Touching a trail tells the user who left this trail and allows them to follow this person.

This system of communicational links exists parallel to the system of paths. It is subject to very fast changes compared to the slow changes of the traffic system. Two categories of variables can be distinguished in Placeworld: the slow changing architectural structure including the path system and on the other hand the very flexible and fast changing flow of communication and traffic. These variables are interrelated (e.g. the disappearance of non-visited places) but mainly the fast changing variables are depending on the slow changing ones. The traffic flow and the resulting communication is determined by the network of paths and the architectural structure. This path-system of embodied traffic and communication seems to be an appropriate metaphor to represent different degrees of activity, interaction and communication.

Controlling the Placeworld

An access model is implemented as an underlying control structure to put some rules concerning the interaction of visitors, the utilisation of the space etc. in force. It provides a very flexible structure to keep track of the ownership of individual cylinders and prevent unauthorised modification by other visitors. The

owner of a cylinder (normally the creator) can specify certain restrictions that will either prevent any modification by other visitors, notify the owner when someone tries to modify the cylinder, or open it for collaborative modification etc. The access model also cares for the zoning of the Placeworld environment and allows the creation of cylinders only in appropriate areas.

The possibilities of the access model can largely be extended and are especially important when file or application access is in question.

In the Placeworld installation that is shown in Jönköping the access model is implemented to an extent that gives a first idea of what is possible with this structure. The implementation takes into account that the average time visitors will be able to spend in front of the Placeworld will be fairly short.

The Placeworld environment could be used to examine how a first settlement in a vague area takes place and how it develops over time. This can be regarded in terms of urban architecture and in terms of social interaction. The resulting structures can be compared to real world phenomena to examine the behaviour and the development of virtual spaces and communities in relation to real world environments. As the virtual environment is not subject to topological facts etc., it is an ideal abstraction of normal space. By defining certain conditions that govern the virtual environment it could be used as a simulation of settlement and interaction. In a short-term perspective, it is interesting to see to what extent a co-operation of different people takes place and what is the result of it.

In this chapter we have considered the design of the overall interface presented to users of Placeworld. We have shown how the same interface can support movement between different styled electronic environments and how the interface can be contextually aware. The following set of closely related chapters focus on the technical development of the electronic environment and the ways in which the environment has been constructed.

Chapter 5: Placeworld Technical Realisation

A number of diverse software technologies have gone into the realisation of the Placeworld. These different techniques are described in detail in the following three subchapters. They include descriptions of :

- the underlying visualisation system, with attention to the aspects and features which have made this system particularly appropriate to the task at hand
- the sets of classes used in the realisation of the Placeworld and the features they support
- the novel access control model required by an advanced, shared virtual environment of this nature.

Each of these different topics are considered in the related chapters that follow. To show the strong links between these chapters we have numbered them Chapter 5a, 5b and 5c.

Chapter 5a: Architecture of the DEVA system: addressing the issues of 'scale' in virtual reality and electronic landscapes

Steve Pettifer, Jon Cook, James Marsh, Adrian West
Advanced Interfaces Group, The University of Manchester

Over the three years of the eSCAPE project, the Deva VR system has matured from its initial form as a 'demonstrator of concept' to a fully-fledged and realistic platform able to support sophisticated virtual environments. Using MAVERIK as its rendering and spatial management kernel and providing its own object management and distribution system, we have demonstrated with Deva a number of features untenable with alternative systems. In this chapter we describe the third and latest incarnation of the Deva system, concentrating on two aspects of its design that enable it to deal with large numbers of objects, environments, users and applications.

In this chapter, then, we are dealing with the provision of the fundamental functionality for 'Virtual Realities'. This is low-level computer science, rather than, say, the more socially informed work we have undertaken in other aspects of the project. Virtual Reality as a general field seems to promise a lot; it is easy to envision complex shared Virtual Environments (VEs) in which people at different geographical locations communicate and work together within CAD, Engineering or Social contexts. This vision is tantalising and must surely bring some benefit if it can be realised. Yet, building VR applications of any sophistication is hard work, as evidenced by the lack of compelling examples of industrial strength demonstrations of this technology in serious use.

Until recently technological limitations were the clear brake on progress for VEs (the graphics challenge is perhaps the clearest example). Beyond the technology the key task is that of writing software. The scale and complexity of this task is often underestimated, and it is here where we believe that the major problems lie. We do not at this time have adequate frameworks to simplify the task of implementing virtual worlds. Today, a person wishing to implement a challenging VE application has two broad options, as evidenced in current demonstrations. The most sophisticated VEs are usually bespoke applications constructed from the graphics layer upwards; this is a substantial undertaking. The alternative is to use an off-the-shelf VE builder. These fall into different categories, (for example, VRML[1] builders at the lower end, up to VE systems

[1] G. Bell, A. Parisi, and M. Pesce. VRML, The Virtual Reality Modelling Language. International Standard ISO/IEC 772-1:1997, http://www.vrml.org/Specifications

such as DIVE[1], Ubi-et-Orbi [2] and MASSIVE[3] at the more ambitious extreme). Excellent though such offerings are, they inevitably impose a particular view of how a VE should be structured. Perhaps it is too early in the state of the art to know how best to do this without unduly limiting what can be achieved with them.

The software support challenge to facilitate large-scale VE applications is, we feel, twofold. Firstly to find techniques and algorithms to address specific needs in VEs. Collision detection, parallelism, distribution, synchronisation, navigation and so forth all require work of this kind. Secondly and perhaps rather harder, it is to find frameworks that allow all the parts to be put together in 'flexible yet powerful' ways. The rather trite nature of such a statement belies the difficulty of quantifying that task.

Finding the 'right' framework is particularly difficult in the case of VR, since it brings together a number of complex technical issues, and binds them with real time constraints at the social/perceptual interface. A desirable approach for the necessary flexibility to experiment is to build as little into the system as possible, so the system provides a set of mechanisms and policies and default behaviours that can be unplugged and tailored at each level. Several technologies exist for providing such general purpose brokering/connection between applications[4], however in our experience and that of others they have proven to be too heavyweight and can neither meet the real-time constraints of VR nor provide the scalability required to move beyond trivial VEs.

The issue of scale is an important one; simple small-scale VEs that do not challenge today's hardware can be constructed by any number of means. Building large-scale complex applications raises a number of challenges. For shared VEs these relate, broadly speaking, to the following areas: number of objects; complexity of behaviour required of these objects; complexity of individual rendering techniques; number and geographical distribution of simultaneous users; and number of co-existing and interacting applications.

One technique for addressing the issue of scale arises from the observation that for a shared VE there is a natural distinction between the individual perceptions of

---

1 C. Carlsson and O. Hagsand. DIVE - a multi-user virtual reality system. In Proceedings of the 1993 IEEE Virtual Reality Annual International Symposium, pages 394-400, Los Alamitos, CA, 1993. IEEE Computer Society Press

2 Y. Fabre, G. Pitel, L. Soubrevilla, E. Marchand, T. Geraud, and A. Demaille. An asynchronous architecture to manage communication, display and user interaction in distributed virtual environments. In Proceedings of Eurographics Virtual Environment 2000, Amsterdam, June 2000

3 C. M. Greenhalgh. Large Scale Collaborative Virtual Environments. PhD thesis, The University of Nottingham, 1997.

4 M. Deriggi, M. M. Kuba, A. C. Sementille, S. G. Santos, C. Kirner, and J. R. F. Brega. CORBA platform as support for distributed virtual environments. In Proceedings of the 6th IEEE Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises. IEEE Computer Society Press, 1997

S. Deihl. Towards lean and open multi-user technologies. In Proceedings of the International Symposium on Internet Technology (ISIT'98), Houston, USA, Mar. 1998

the VE and what is actually `going on' within it [1]. That is, between the distribution and simulation of application behaviours, and the task of presenting a coherent perception of the VE to each user. This is similar to the traditional philosophical distinction between subjective perception of an underlying objective reality. The reason that this is useful is that in VR it separates out the challenging graphics/interaction tasks from the semantics of the underlying world simulation. The distinction also legitimises efforts to make a perceptually smooth presentation of the world in the light of fundamental networking limitations. Architecturally we have addressed these two aspects of the VE (perception/reality) separately. First, rendering and spatial management seem to need special treatment that is different from current approaches to graphics toolkits. The MAVERIK system[2] aims to address these issues. The reason, hinted at earlier, is that no one structure or representation is appropriate for the range of interactive applications envisaged, so no one graphical representation can be selected as the `internal' structure of the VE system. This is as much a performance issue as programming elegance, for key graphics optimisations are highly application specific, and are generally unavailable when the application must export its representation into the VR system.

The more challenging task, namely the simulation of the underlying reality regarded separately from its graphical representation, comprises two core services:

1) An execution environment: a means by which the application and user interactions can be integrated in a single run-time environment. Operating systems history suggests that one should design for a number of applications to either run simultaneously in the same environment, or to allow the user to move from one environment to another seamlessly
2) Distribution: a means of distributing the actions of the user and the behaviour of the application across a wide area network.

Integrating multiple applications may be simple where each application effectively conforms to a particular ‘world view’. This is to say for example that a certain event model is used throughout, that interaction takes place in one of a set number of ways, and that objects are one of a set number of basic ‘types’ with essentially only superficial differences. However our experience of building non-trivial VEs suggests that this is too inflexible – the one-size-fits-all approach seems too impoverished for real tasks, and our MAVERIK system has proven to be an effective way of avoiding this problem.

---

1 P. M. Sharkey, D. J. Roberts, P. D. Sandoz, and R. Cruse. A single user perception model for multi-user distributed virtual environments. In Proceedings of International Workshop on Collaborative Virtual Environments, Nottingham, UK, June 1996

2 R. Hubbold, J. Cook, M. Keates, S. Gibson, A. Murta, A. West, T. Howard, and S. Pettifer. GNU/MAVERIK: a micro-kernel for large-scale virtual environments. In Proceedings of ACM VRST99, London, UK, Dec. 1999

Providing a mechanism that allows any disparate set of navigation techniques, rendering routines and so on to be present in a single environment is desirable, but the conflicts, inconsistencies etc. that this would present to the user make this unrealistic if it is not moderated. For example, different navigation metaphors are used in data visualisation, CAD design review, and interactive art works. A framework therefore needs to provide a way of managing these different kinds of techniques, which as well as enabling integration at a technical level provides some means for users to comprehend what is occurring in the environment, and what possibilities it offers as they move from environment to environment, application to application.

We have partitioned the VR architecture task into three parts. The first, graphics and spatial management—the users perception of the environment and its rendering using the MAVERIK system, we deal with elsewhere though the integration of that component will concern us here too. The second and third parts, namely the execution and distribution facilities, we have argued are the central concerns for managing the underlying world model. In this chapter we describe the concrete implementation of these ideas in a VR system called Deva.

Distribution and communications

The programming model employed by the Deva system is one of communicating ‘entities’ which can represent objects in the Virtual Environment, the properties of the environment itself, or alternatively abstract programming concepts that have no direct representation to the inhabitant of the environment. These entities are coarse grained programming objects, exporting a number of ‘methods’ that can be called by other objects, and implementing these internally using optimised imperative code (currently written in C++). In this section we describe the distribution architecture that is employed by Deva that enables transparent and lightweight communication between entities distributed around the system. A detailed description of the makeup of an entity is presented in a later section; for now we concentrate on the techniques used to rapidly route messages between entities around the system.

The distribution architecture

Deva is logically a client/server architecture, which to a first approximation provides a single definitive locus of control for the Virtual Environment using its server component, with ‘mirrors’ of the entities being maintained in each client process. Behind the scenes, however, Deva pragmatically manages the delegation of control dynamically to the most appropriate parts of the system, thereby achieving the highest fidelity of perceptual and causal coherency attainable for the application at hand.

A diagram of distribution architecture is given in Figure 1. The ‘server’ is in fact a cluster of processors running identical processes called ‘server nodes’ that together form a single multi-threaded parallel virtual machine capable of processing large numbers of entities. The intention is that the server provides a computing resource for multiple virtual environments, and maintains a far heavier processing load than any one user’s client could manage at any one time. A networking layer provides lightweight position independent messaging between entities. Entities are created in and managed by the server node processes, and client processes—such as a visualiser or user application—connect to the server to interact with and obtain state information about the entities.

The server is persistent: it remains alive processing any entities regardless of whether or not any clients are connected. Administrative tools exist to simplify the startup, monitoring and shutdown of the parallel server.

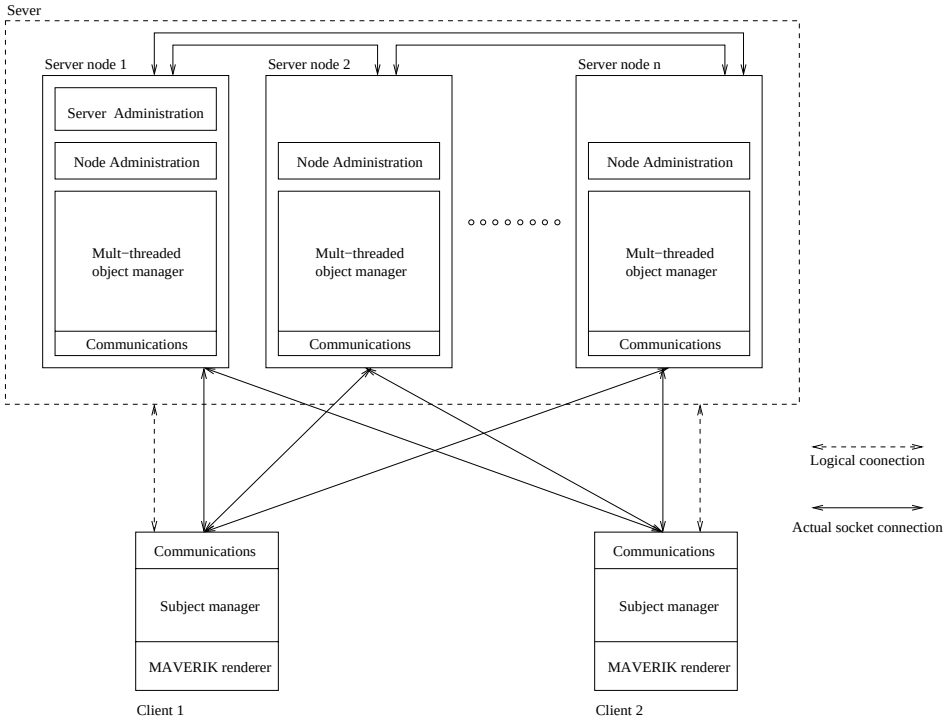


Figure 1: Overall architecture of the Deva system

Creating and addressing entities

Each server consists of a (large) fixed number, *M*, of virtual servers. The *M* virtual servers are trivially mapped using a lookup table to the *N* server node processes which comprise the server. This conversion takes place at a low-level within the system and essentially hides the configuration of the server.

An arbitrary virtual server is chosen to create and manage each entity (currently a random virtual server is chosen but this selection process could take into account loading factors).

Each server node potentially manages multiple entities. Each entity is assigned a unique 'pool ID'—an offset into the list of entities managed by a given server node.

The location of an entity in the server is uniquely defined by the virtual server and pool IDs. This pool ID is not strictly necessary and is provided for efficiency only since each entity has a unique name that can be searched for in the list of entities managed by the server node.

When an entity is created a hash function is applied to the entity's name to obtain a second virtual server. This virtual server manages the *name* of that entity, i.e. it definitively knows the virtual server of where the entity is actually located and its pool ID. The name of the entity and the entity itself are managed by separate virtual servers.

The same hash function is used throughout the system allowing any Deva process to obtain the location of a named entity. This data is cached for future use.

Entity name to location lookup, while a lightweight process, is a central and frequent task in a distributed system. A scheme where this load is spread equally across all servers nodes is therefore advantageous.

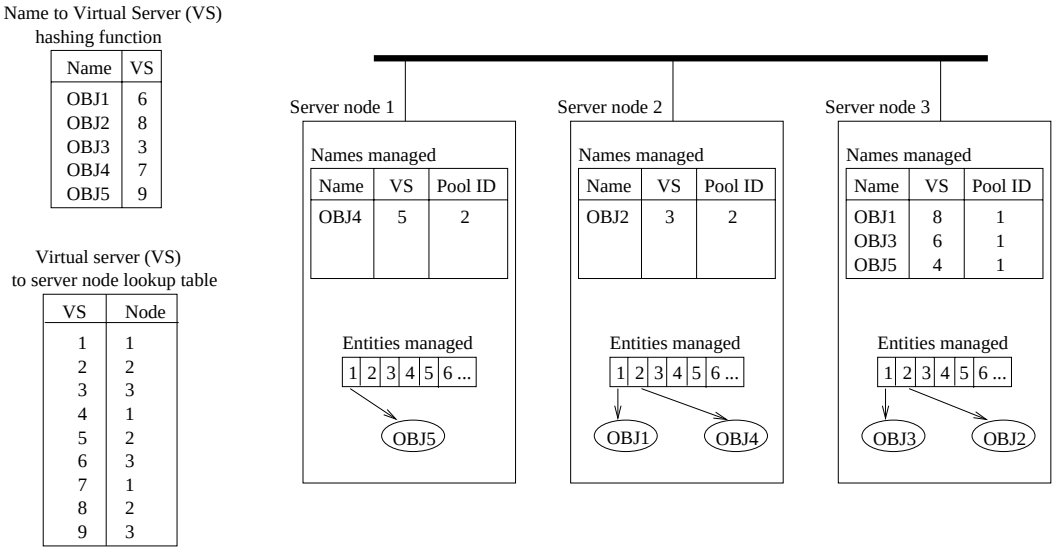


Figure 2: Distributing entities and names across the system

Figure 2 shows an example distribution of entities and name management across server nodes.

Entity migration

The main advantage of the addressing mechanism is that it allows entities to dynamically migrate across server nodes to help balance processing load. When an entity moves it only needs to inform its name manager of its new location. Deva processes—which now contains out of date cached data—will receive an error the next time they try to communicate with the entity at its old location. This error is trapped internally and the new location of the entity is obtained from the name manager; the originator of the communications with the entity is oblivious to the migration. The name manager can always be relied upon to know the correct information and its location is trivially obtained.

Server reconfiguration

It is also possible to migrate the names managed by a given virtual server onto a different server node by updating the virtual server to server node lookup tables in every client and server process.

The migration of both entities and name management allows server nodes to be dynamically added and removed from a running server.

Networking protocol

Currently standard TCP/IP point to point socket communication is employed although some work has been undertaken investigating multi-cast, since for local area networks at least it promises improvements in performance for our application.

Although not a strict requirement, the communications strategy is based upon the assumption that inter server node communication is fast compared to server/client communication. For example, the server nodes are connected via a dedicated network or protected to some extent from superfluous traffic by a bridge; while clients connect to the servers via a high-traffic shared LAN or modem connection.

Synchronising the behaviour of entities

The distribution of semantics or `behaviour' in shared VEs is a particularly difficult issue. The ‘ideal’ solution would be to describe the required behaviour in a single location and to have it *instantaneously* sent to all participants (a pure ‘client/server’ approach). Current network technology is far too limited to make this solution feasible. An alternative is to replicate behaviour locally for each participant (a pure ‘peer-to-peer’ approach). Though the favoured solution in many VE systems this however introduces problems of synchronisation.

The approach taken in Deva is a hybrid solution, part way between client/server and peer-to-peer. The model is one of an objective and subjective reality; the former being located on the server, the latter being represented on each client. Each user interacts with the objective reality via their own subjective view. This introduces the idea that each entity in the virtual environment has two definitions, which may differ significantly in their semantics. Our usage of the term `subjective' is not intended to imply that each user experiences an entirely different VE (this would hardly count as a shared experience)[1]. Rather, we argue that *an amount* of the user's experience may be decoupled from the `objective' behaviour of the world without disturbing consequences. Given that absolute synchronisation is in any case impossible, it is our contention that so long as the length of the delay is not *too* large, and causal events occur in the correct order, it is possible to accept a degree of subjectivity without it affecting the users' understanding of the semantics of the application. Under these circumstances, given no other frame of reference the users are unlikely to even realise they are not receiving the same view of the world as each other. In general it would therefore be useful to differentiate between what is 'actually' happening in a VE and what the users perceive. In this way it is possible to optimise the manner in which information is transferred to minimise causal discrepancies.

In the Deva system this notion is implemented by describing an entity as comprising of a single 'object' and a number of 'subjects', the former part being the 'objective non perceived' aspect of the entity and the latter being a collection of the 'subjective perceivable' components (e.g. its visible, audible, haptic parts). An alternative view of this distinction is that the 'object' represents what an entity does, whilst its 'subjects' represent what the entity looks/ sounds/feels like when it is doing it.

Communication between an object and its subjects can be implemented using whatever high-level 'vocabulary' is appropriate. In this way the system minimises the need for strict synchronisation whilst maximising the accuracy of causally important events. One example of this is the use of Twines[2]. Twines are lightweight parametric curves that are used to `smooth out' spatially and temporally the discrete updates to the visible aspects (e.g. position and rotation) of appropriate entities by interpolation thus increasing their 'plausibility' by eliminating disconcerting jumps. We are also currently investigating an infrastructure for providing 'quality of service' based on balancing such smoothing against frequency of updates depending on the perceived importance of various events.

The flexible nature of the communications is also advantageous for subject to object communications; for example when a user 'grabs' and moves an entity. Such manipulation with a classic client/server architecture would involve a round

---

1 S. Pettifer and A. West. Subjectivity and the relaxing of synchronisation in networked virtual environments. In Proceedings of ACM VRST99, London, UK, Dec. 1999

2 J. Marsh, S. Pettifer, and A. West. A technique for maintaining continuity of perception in networked virtual environments. In Proceedings of UKVRSIG99, Salford, UK, Sept. 1999

trip: the event is sent from the client to the server to be processed and then the effects of the event are then transmitted back to the client to be visualised. With the commonly available network technologies we are targeting for the client/server connections such a round trip would introduce an unacceptable lag. It is perceptually important for the cause and effect of the manipulation to be as tightly coupled as possible.

With the separation of subject and object, changes caused by the manipulation can be immediately perceived by the subject, with a user-definable policy function being used to determine when the object is updated. For example, an arbitrary fraction of the changes (say, one every fifth of a second) are transmitted back from the subject to the object and thus onto other visualisers connected to the server. Alternatively, a change in position is only updated to the object when the subject has moved a certain distance from its previously synchronised location.

The goal is to ensure that the entity behaves ‘correctly/optimally’ on the visualiser performing the manipulation and also that it behaves `plausibly/acceptably' on any other visualiser connected to the cluster (while accepting that latencies and bandwidth issues rule out strict and absolute synchronisation in such a distributed system).

Defining the behaviour of entities

Commonly, VR systems define the behaviour of an entity by attaching pieces of code, often written in Java, Tcl or some other scripting language. With little specific architectural support however, it is often laborious to code and difficult to ensure consistency. To some extent this situation may be improved by taking an object-oriented (OO) approach; objects with similar characteristics can inherit from common base classes. There are however no viable OO languages with the necessary support for dynamically changing the inheritance graph {\em at runtime} when, for example, an entity moves between worlds (such as from one with the concept of gravity to one without).

We can identify at least four conceptual sources of behaviour for each individual entity within a VE:

1. **Behaviour innate to an entity.** Entities each have their own particular role within a VE, for example the ability of a stopwatch to record elapsed time.

2. **Behaviour common to a range of entities**. Often many entities share some aspects of their behaviour through being of a similar type. For example each pawn on a chessboard is subject to the same detailed rules governing movement; these rules do not apply in the same way to other pieces which have their own restrictions. However, all the pieces have in common the rules determining whether they may be ‘captured’; a notion that is specific to chess pieces but not necessarily to other objects in the same environment. Thus a hierarchy of common groups can be determined.
3. **Behaviour common to all entities in a particular world**. We generally consider gravity to be a phenomena associated with everything found in the world around us rather than as a property of each individual object. This also applies to social constructs such as `monetary' value. An earlier version of Deva first introduced the notion of environments imbuing behaviour the entities it contains.
4. **Behaviour that is dynamically required at runtime**. If an entity becomes inhabited by a user for example, it will behave differently---being controlled by a navigation device and so forth---to when uninhabited. Similarly an entity that is ‘set alight’ will suddenly have properties previously unavailable to it (for example, being able to set fire to neighbouring entities or raising the temperature of an environment).

Components

Our solution to the problem of merging the various sources of behaviour which comprise an entity is to use ‘components’. In Deva a component is a collection of methods and attributes relating to a single concept that can be attached or detached from an entity at runtime.

The methods and attributes which comprise the component are divided into a single ‘object’ and multiple ‘subjects’ as outlined in the previous section. While typically a component will contain both parts, some components are entirely abstract and have no directly perceivable representation in the virtual environment, i.e. they have no ‘subject’ part.

The researcher trying out new low-level ideas is free to write components directly in C or C++ that interface to the VR kernel at whatever level is appropriate. More general users are free to use the library of existing components to construct entities without concerning themselves with their implementation. We speculate also that this component system has the added benefit of being usable from an immersive ‘graphical user interface’ allowing the user to ‘mix and match’ components at runtime.

In order to facilitate efficiently moving entities between worlds with different behaviours, they contain two lists of components, one inherited from the environment, and the other containing its own innate behaviour. Traditional object-oriented inheritance is supported through components being able to load

other components when they are initialised. The order that the components are searched enforces the correct precedence.

When a method is called upon an entity, the two lists are searched in strict order. First of all the list of components given to the entity by the environment is scanned for methods marked as being ‘enforced’. If one with the correct name is found, then this is called. Enforcing methods allows an environment to ensure all entity contain a particular method that cannot be overridden. Next, the entity's innate components are searched. If the method is still not found then the environment components are searched again for methods marked as ‘imbued’. These are methods given to the entity by the environment but which can then be overridden by the entity itself. For example, an environment may enforce the notion of ‘solid objects’ upon all its entities (‘you can't pass through walls’). It may also ‘imbue’ all entities with a concept of mass with a default value estimated based on its volume, but which the entity is free to over-ride should it prefer.

Methods and Filters

The strict order in which components get searched for methods leads to another useful concept, that of the filter. As well as methods just returning results, they can be declared to be filters. These are allowed to return a new method call as their result which then continues to propagate along the component lists. This is useful in a number of situations. Say, for example, it was necessary to constrain an entity to a plane. The entity inherits the imbued method `setXYZ` from its `ThreeDSpace` environment. In this case the entity would define a component itself that defines `setXYZ` as a filter. Since this would be found before the imbued method, the entity's filter is able to constrain the coordinates to a given plane before returning a new `setXYZ` message with the new coordinates that then propagates through to the environment imbued `setXYZ` method. Another example would be storing an entity's state prior to migrating the entity to another server node process or prior to a complete shutdown of the system. Each component defines a `snapshot` filter that adds any state variables it possesses to the input message and returns a new `snapshot` message. The final component then returns a `restore` message containing all the variables that is sent to the new entity. The new entity contains a `restore` filter in each component that originally stored variables which then unpacks the message.

Continuous behaviour is supported by permitting components to define a method called `activity` which is polled intermittently. Activities are not subject to the usual calling precedence, but always polled if defined.

Figure 3 shows how components are combined to form a logical entity.

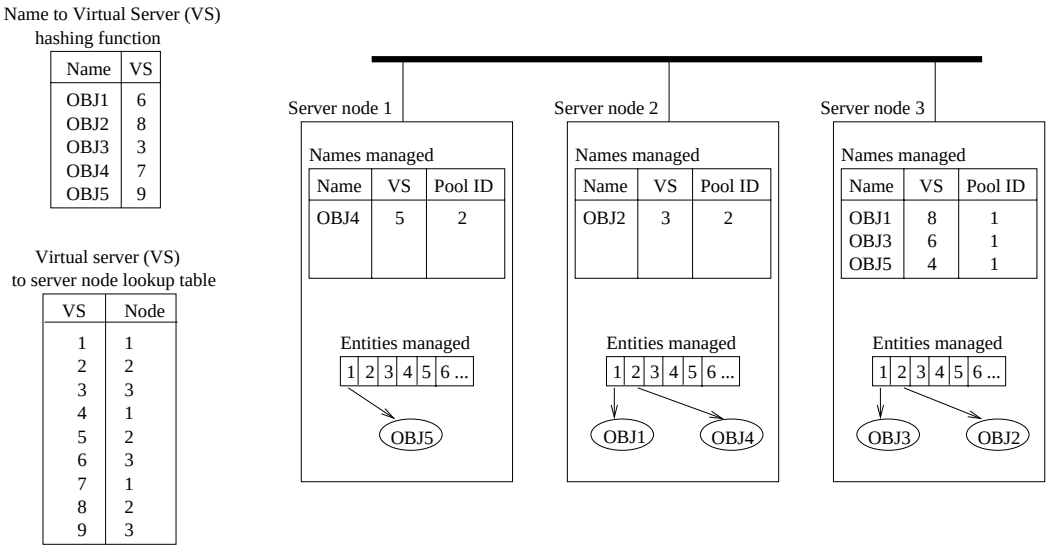


Figure 3: The structure of a logical entity

This chapter has provided an overview of the Deva architecture used to support Placeworld and the core technical component exercised in the construction of this environment. The developed environment also required some new mechanisms to be added to Deva. One such extension was the addition of a new set of Maverik classes that implemented specialised graphics facilities for Placeworld. These are described in the following chapter.

Chapter 5b: Implementation of the Placeworld - MAVERIK Classes

Simon Gibson, Jon Cook, Steve Pettifer, James Marsh
The University of Manchester

One of the main aims of the physical electronic landscape has been to allow the contents of the virtual space to be modified or enhanced by its inhabitants. At a 'social' level, this is an issue of providing the appropriate metaphors and tools to allow the editing of the environment itself using objects that already exist in the world; our experiments with the access model described previously aim to address some of the fundamental problems associated with the collaborative reshaping of a VE. At a more technological level, the issue is one of systems architecture and integration. Traditional virtual reality systems have a fixed view of the type of objects that reside in their environments. Whether these be of a single type (e.g. 'polygonal mesh') or instances of a number of base classes (e.g. 'camera', 'person', 'animated object') certain unwanted restrictions are placed upon these entities (for example, a system that has sophisticated support for mesh objects may not have equal support for particle systems, or radiosity solutions and so on).

One approach to this problem is to attempt to provide 'base classes' for every conceivable type of object that could be coded. Clearly this is an unrealistic aim. Our solution to the problem is to build as little 'into the system' as possible, and instead to provide users with support that makes coding their own objects 'from the ground up' a relatively trivial matter. This includes the provision of the 'behaviour' of the objects as well as the routines that render them into the environment (the distinction between rendering routine and behaviour is however rather blurred, as will be demonstrated later in this chapter).

Specifically, this is achieved in the Deva distributed environment by utilising MAVERIK classes; encapsulated information and functionality about the rendering and spatial management of an area of space and/or its contents. A detailed description of the MAVERIK kernel is provided in the freely available installation available on the web (http://aig.cs.man.ac.uk/systems/MAVERIK)

The appearance of Placeworld has been informed by artistic vision and graphic design. As a 'public place' such aesthetic issues are important, and help to set the environment aside from other similar worlds that use primarily mesh or scene-graph based rendering packages. In this chapter we describe the new MAVERIK classes that have been written for Placeworld.

Avatars

MAVERIK provides a default avatar; a simple polygonal representation of a human male consisting of an invisible skeleton and a number of simple geometric shapes built using AC3D and connected to that skeleton at appropriate points. The avatar can either be positioned accurately by the MAVERIK programmer, or can be used in a more 'interactive' mode whereby the position over a number of frames is examined, and a simple 'walking' motion effected on that basis using Twines[1] to animate the movement of the limbs. Further, a limited form of inverse kinematics may be activated to cause the avatar's arms to be positioned 'correctly' to fit a hand position. Thus the avatar may be made to gesture or point to objects in the environment in a plausible manner.

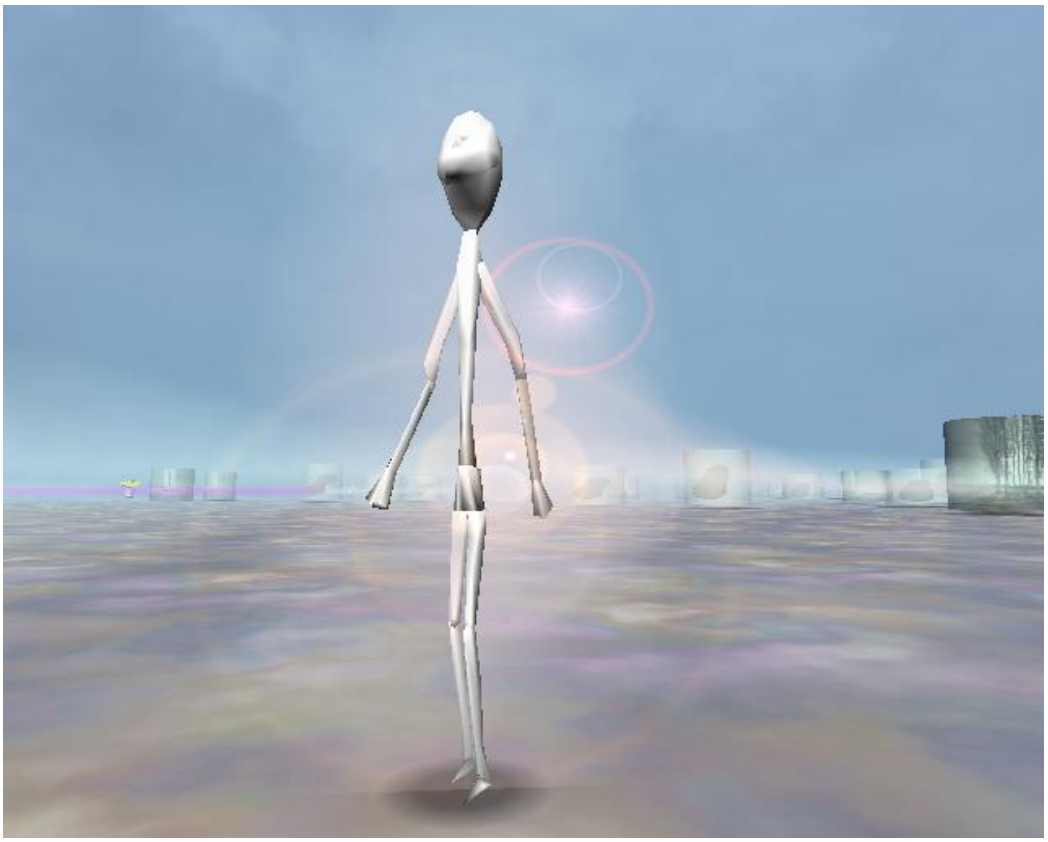


Figure 1: The Placeworld 'stickman' avatar

For Placeworld, a more esoteric avatar was implemented. This consists of two phases: stripping and replacing the main body objects of the default avatar with new 'skin', and the addition of a glowing trail to represent the path of the avatar in the environment over a period of time.

1 J. Marsh, S. Pettifer, and A. West. A technique for maintaining continuity of perception in networked virtual environments. In Proceedings of UKVRSIG99, Salford, UK, Sept. 1999.

The new body was designed by a graphic designer, based loosely on figures from H.S. Lowry paintings (Figure 1). Once positioned in the environment, the avatar's location is sampled once every 10 seconds, and a trail is constructed using a Twine (a parametric cubic curve). This allows other users to see areas of the environment that are popular, and also to 'chase' other avatars (qv. Nuzzle Afar[1]).

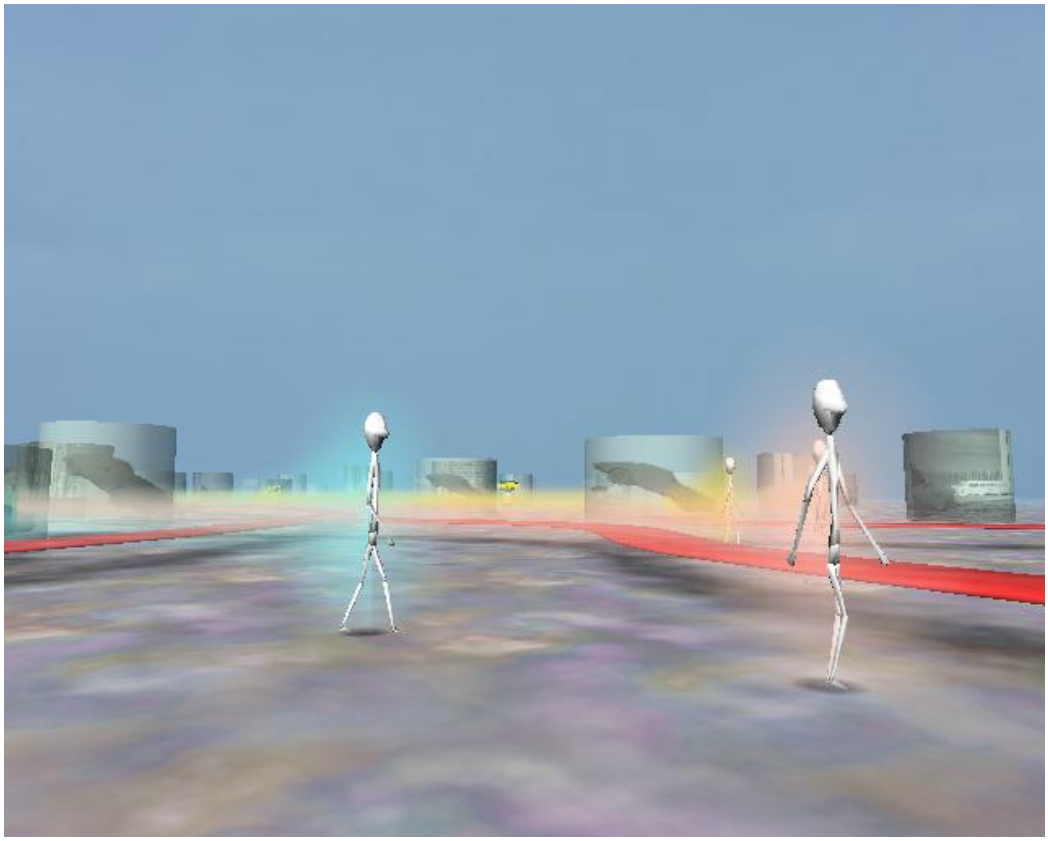


Figure 2: Two avatars leaving coloured trails

The trails themselves are rendered using semi-transparent textured polygons with the alpha-value being modulated along the length of the trail to cause it to 'fade out' gently towards its tail.

A small shadow polygon is placed underneath each avatar and blended in with the ground texture.

Cylinders

The cylinders that form the main body of Placeworld are built using modified MAVERIK cylinder classes. Given the large number of cylinders that will

1 Fujihata, Blunck, Vincent, 1999: "Nuzzle Afar" in *escape deliverable 5.1,* Lancaster

eventually populate Placeworld, a number of optimisations have been implemented.

The cylinder itself uses the built-in distance based level of detail routines to reduce the number of polygons rendered for each frame. Cylinders are either rendered using a semi-transparent panoramic texture (qv. Last year's TIC demonstrator 'PLACE' objects) or using an SMS Silhouette as described below. Each cylinder renders a blended shadow underneath it, giving the impression of being slightly raised off the groundplane.

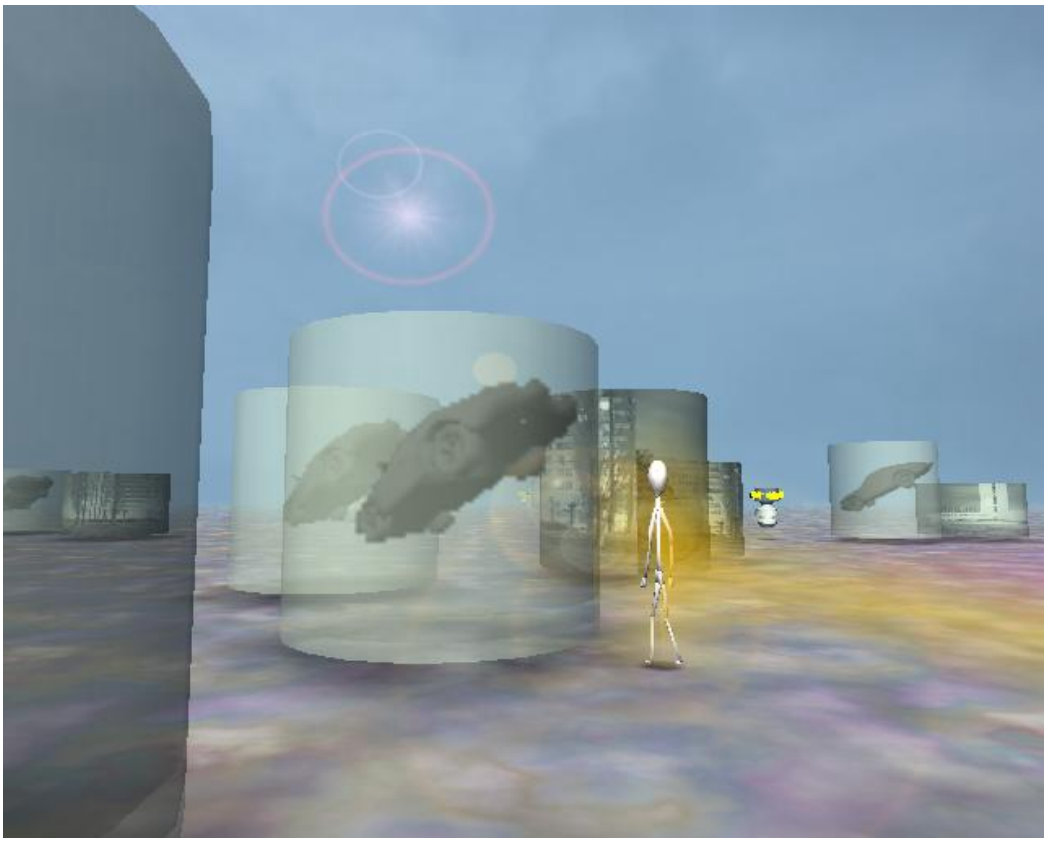


Figure 3: An avatar approaches a cylinder containing a silhouetted car model

SMS Silhouettes

Each cylinder in Placeworld potentially represents an entire new virtual environment. This may be as simple as an enclosed space containing a number of geometric primitives, or, exploiting the Deva system's ability to manage multiple simultaneous applications, could be an entire, self-contained, alternative VE such as the Distributed Legible City ('DLC', see last year's deliverables). Each place cylinder is rendered semi-transparent, to enable users of the main Placeworld environment to see what is inside before entering. Clearly, for an environment of the complexity of the DLC, a brute force rendering of the entirety of the

environment is not possible in real-time. To overcome this problem, an image-based rendering technique is employed to provide impostor objects that represent the contents of the internal VE.

The sub-environment is contained within its own MAVERIK Spatial Management System (SMS), and a single silhouette texture is constructed for each environment. The texture stores an image-based representation of the contents of the SMS when viewed from a large number of directions. The texture is constructed by rendering the SMS from each direction into the back-buffer, reading the buffer, and storing the image in the texture map. By reading the depth buffer as well, texels not covered by the SMS contents can be found, and rendered transparently using alpha-blending. A billboarded polygon (i.e. one that always rotates to face the viewer) is used to render the silhouette. Depending upon the angle at which the Silhouette is being viewed, an appropriate sub-texture is selected from the silhouette image, and drawn on the billboard polygon.

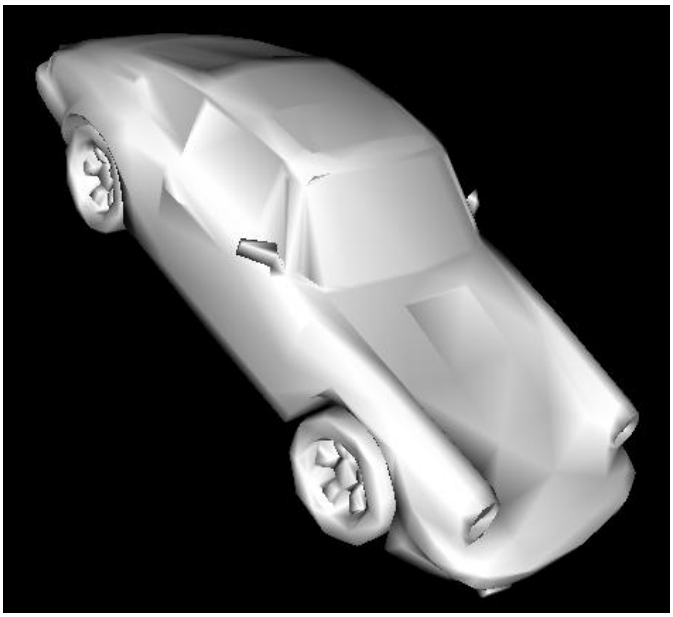


Figure 4: The original model

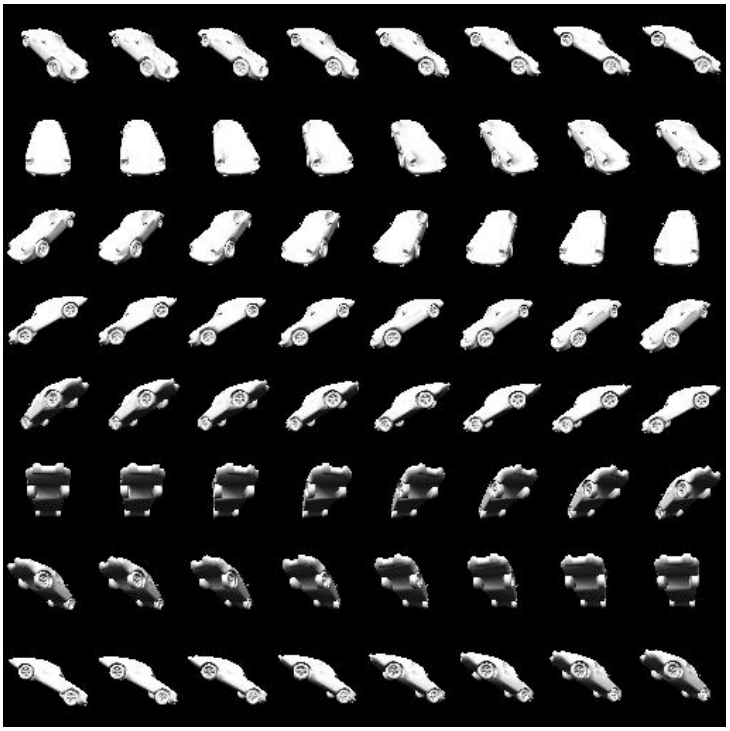


Figure 5: Multiple views of the original model captured as a single texture

The effect is that of seeing some 'real' content within each cylinder, without the need to render any significant number of polygons. The number of textures generated and the resolution of those textures can be varied depending on the hardware used to render the final environment; the greater number of textures and the higher the resolution, the more compelling the final effect.

Signposts

Placeworld is populated by a number of 'signpost' objects that are used to indicate areas of interest, to provide information for users generally, and are objects that the inhabitants of the environment can create for themselves. In keeping with the cylindrical simile, the majority of these signpost objects are cylindrical, having 'rotating hoardings' containing information.

The main body of the object is created using the AC3D editor, and a position is nominated around which the rotating text is positioned. The lettering of the text is billboarded so that as it rotates, the 'seam' always faces away from the reader. Thus a continuous seamless stream of text can be generated without any apparent breaks in the rendering (i.e. the user can never see where the text is being generated, nor where it disappears… a subtle effect that would not be possible to achieve for a solid object in the real world!). The text can be of any length, and will repeat indefinitely. As with all other objects in Placeworld, a shadow is generated underneath the signpost and is blended in with the groundplane.

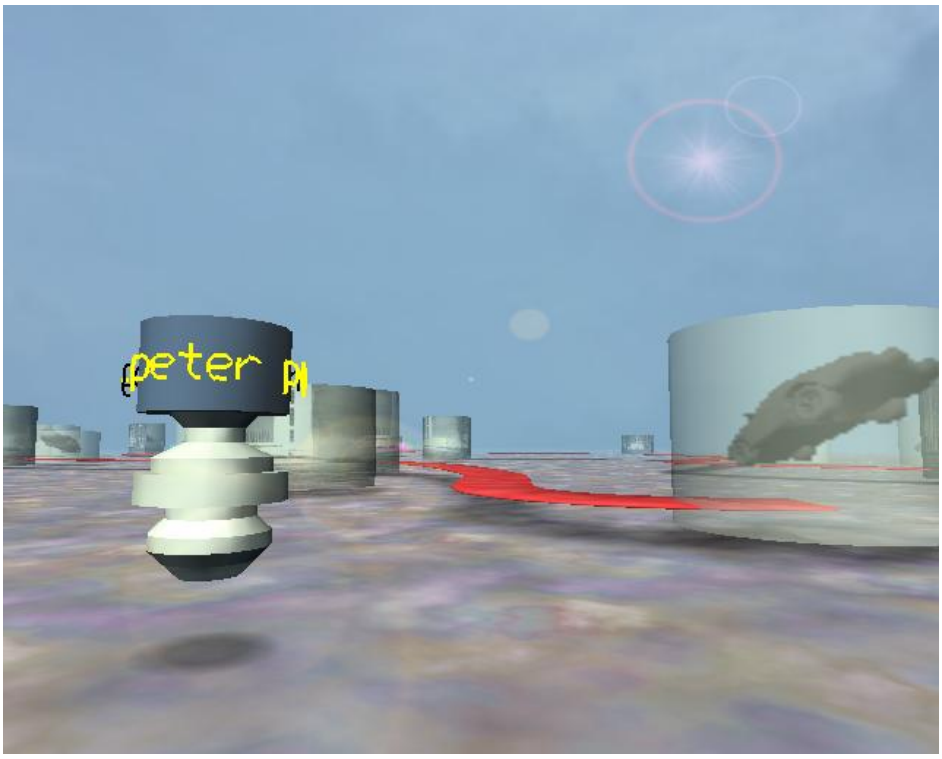


Figure 6: A Placeworld 'signpost'

For effect, the text of the signpost is occasionally flickered, giving the impression of a ‘neon sign’. The entire signpost gently wobbles around its given position, highlighting its free floating nature.

Image-Boards

Image boards are a modification to the sign-post class, and instead of containing rotating text, consist of a sequence of images. These are comparable to real-world billboards/advertisements or to ‘banner adverts’ from web pages. The image sequence can be of any length, and the duration between each image can be specified.

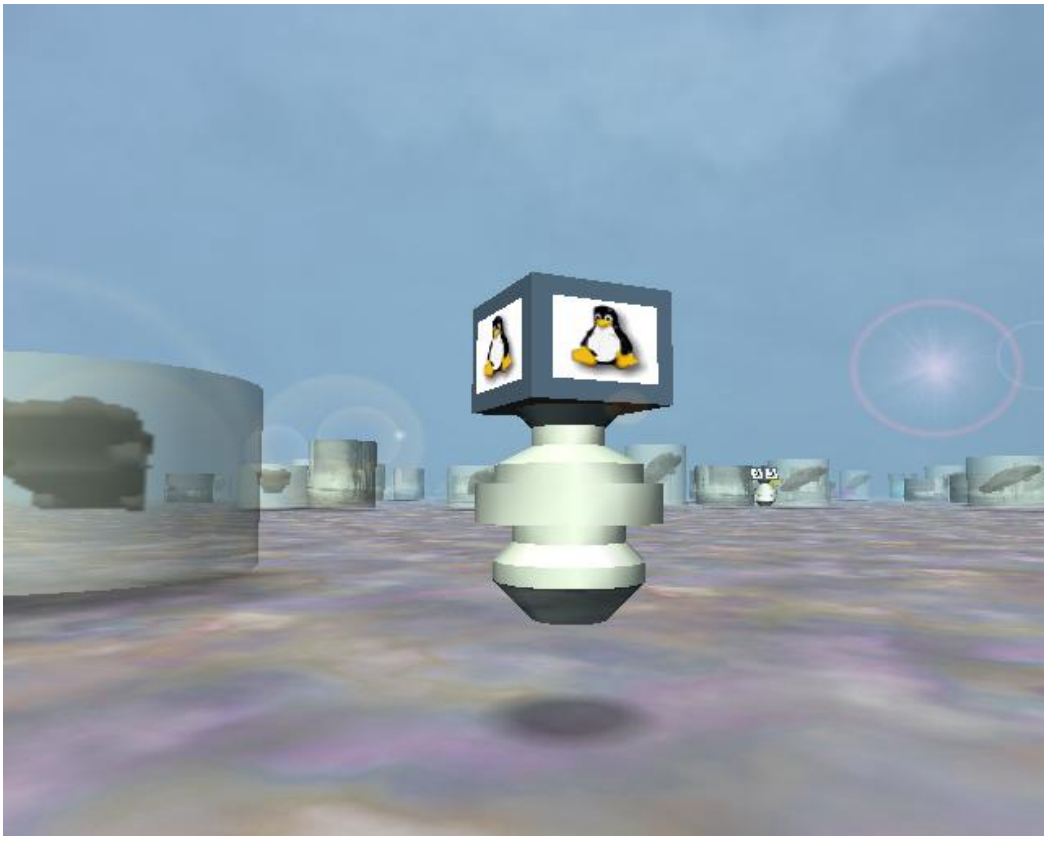


Figure 7: Advertising hoarding

Replicator

Replicators are the sources of objects for the inhabitants of the Placeworld. From these devices, users can obtain new home cylinders, or other artefacts to populate those cylinders. The main body of the replicator is an AC3D object, and any other object created using MAVERIK/Deva can be 'replicated' with this class, whereby it appears inside the replicator object with an animated special effect (there is no actual replication of data; this is purely a visual effect). A simple particle-system is used to render a swirling, glowing effect around the object being replicated. A combination of transparent textures and accumulated intensity values are used to give the glow. The object is also scaled and animated independently in its x,y and z directions to give the impression of 'popping' into existence.

Figure 8: A replicator creates a cube

Lens-Flare

For purely aesthetic effect, the environment contains a lens-flare. This is rendered using a MAVERIK 'frame3' function, in order to guarantee that the lense flare is drawn at the end of each frame. The LenseFlare class allows the specification of the position of the light-source, and multiple flares may be positioned within an environment (we have speculated previously that lense flare, though a mostly artificial phenomenon, may give useful navigation clues by subtly indicating the position of a 'sun' object in the environment thus aiding orientation).

In the implementation of the lense flare, the light-source position is projected onto the image-plane, and the vector from that point which passes through the image-centre is found. Lens-flare textures (generated using Adobe Photoshop) are rendered at appropriate positions along that vector. All rendering is two-dimensional, and is done directly onto the image-plane.

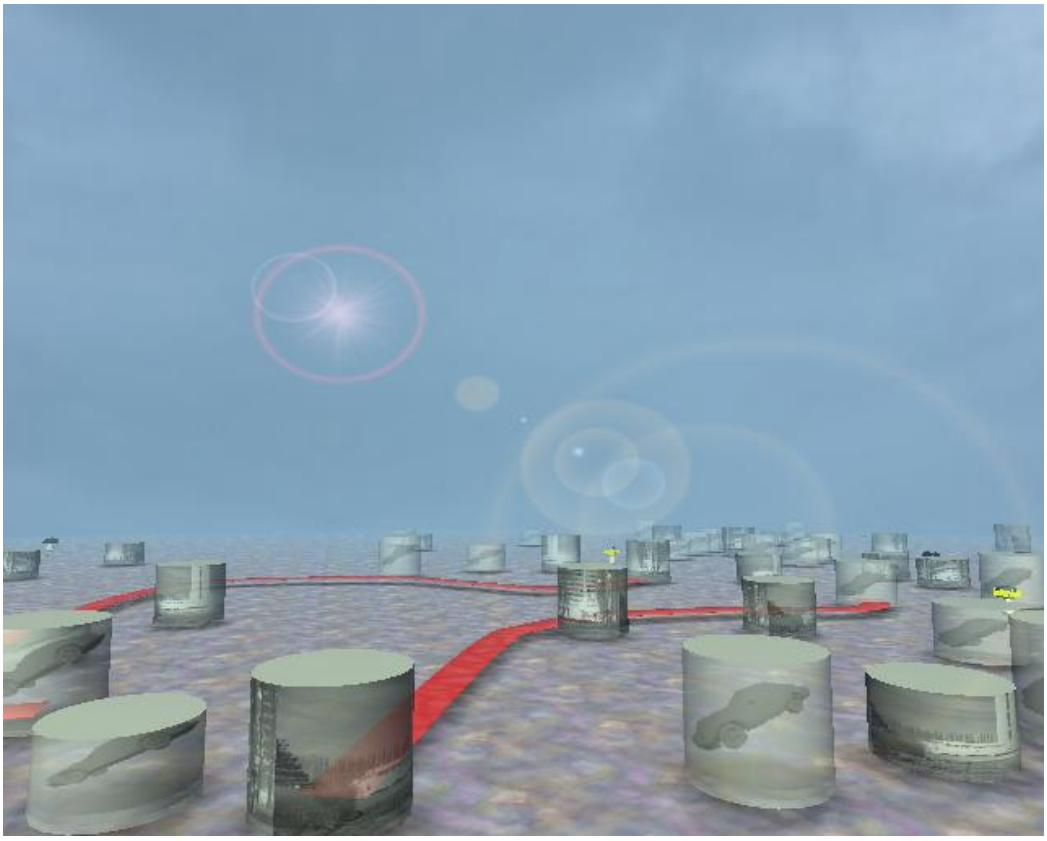


Figure 9: Lenseflare in Placeworld

Sky/Groundplane

The sky is a textured hemi-sphere. The ground is rendered by projecting the vertices of the hemi-sphere onto a horizontal plane, and then rendering. This improves image-quality when per-vertex fogging is used. OpenGL Fog is added to blur the sky/ground boundary.The sky and ground are always rendered so that they are centred at the current viewpoint. This prevents the user walking past the edge of the sky hemisphere whilst avoiding the need to draw a 'to-scale' sky.

The texture-co-ordinates of the ground-plane are changed as the user walks around, to give the impression of movement. Two levels of ground-fog are rendered using semi-transparent textured polygons. Texture coordinates are animated to 'swirl' the fog around.

Icons

Each icon has a semi-transparent texture map, drawn in two-dimensions directly onto the image-plane. The implementer can select the position and size of the icon. Any number of 'hot-spots' may be added to the icon. If the user presses the mouse button when the mouse is over a hot-spot, it is highlighted, and a user-

specified callback is executed. The virtual keyboard is implemented using a single image map and a large number of hotspots with a single callback. Icons are rendered using a frame3 function, so they are drawn after all other objects in the environment.

Figure 10: A 'virtual keyboard' constructed using the icon class

This chapter has presented a description of the nature of the Maverik classes used to implement the key visual components of Placeworld. In addition, the development of Placeworld also required the design and development of a novel access model for virtual environments. In the following chapter we shall consider the design and development of the Placeworld access model.

Chapter 5c: The Placeworld Access Model

Steve Pettifer, Tom Rodden
The University of Manchester, Lancaster University

In this chapter we discuss the development of a new mechanism for allocating and managing the relationships between users and the contents of an electronic landscape. We present a flexible access model designed for this purpose.

The term ‘access model’ refers to the set of mechanisms used by a system to determine the operations that may be carried out on an object by a user. The most common example of an access model in computing is that provided by most mature operating systems to manage files. These techniques can usually be reduced to a three dimensional matrix with users, files and ‘operations’ (such as read, write, execute) along the axes, with entries in the matrix representing ‘grant’ or ‘deny’. Various two-dimensional interpretations of this 3D matrix are called variously ‘access lists’ or ‘capability lists’. These models, however, make a number of assumptions that whilst being entirely valid for the domain in which they were developed mean that existing access models do not work well in electronic landscapes. Essentially, when viewed from the perspective of large scale spaces populated by a diverse set of users and containing objects with rich sets of behaviour two main shortcomings can be:

1. **Limited model of shared use**. Most access models often assume a ‘thin’ hierarchy of classes of user. For example, a typical Unix or WindowsNT system has ‘root’ or ‘administrator’ users that have access to all resources on the system and ‘regular’ users that have access granted through membership of different groups. Though it is possible to extend this system to some extent (e.g. in Unix through the use of ‘set-uid’ or ‘set-gid’ parameters and multiple groups), the system itself provides a three-tier hierarchy as shown in figure X.
2. **Limited object model**. They often assume a small number of object types where the appropriate operations maybe pre-defined and built into the system. For example, on Unix the basic file permissions are ‘Read/Write/Execute’ defined for each of the sets of ‘Owner/Group/Other’. Since in Unix and WindowsNT, processes and other devices and services are often mapped onto ‘virtual files’, the file system’s access model is to some degree applicable to them also. However, there is no provision for extending the types of access available, and the mapping of file to ‘general resource’ is not always a comfortable one.

Given the need for the Placeworld to support the shared evolution of its contents and to develop into a large scale virtual environment containing many users and different types of objects it is clear that a new model is needed. In this chapter we present the development of this model, its technical realisation and describe how it can be applied within the context of a shared environment.

It is worth making a clear distinction between an ‘access model’ and a ‘security model’. The former is first a policy decision, and second an implementation by the system of that policy that enables the moderation of the relationship between a user and the other objects in a system. The latter consists of algorithms that enable the access model to retain its integrity under malicious attack.

Access models focus on the ways in which the availability of resources are managed. In principle, an access model could be a ‘gentleman’s agreement’ between users (though in practice it would be unworkable due to the large number of objects involved if for no other reason). While they can be used to ensure the secrecy of resources they can also promote openness. The access model is the means by which the collective decisions of the nature of the environment are expressed. The ‘security model’ is what enforces the access model. Computer security, with all its associated encryption algorithms, political implications and complex mathematics is a vast topic that we do not intend to consider in depth here. We concentrate instead on describing a novel access model and take into account the issue of security only in as much as it impinges on a prototype implementation of our scheme.

The model we propose aims to provide a mechanism whereby inhabitants of an electronic landscape may describe how they wish entities that they create/own to be related to by other inhabitants. We wish to avoid prescribing the types of operation that may be permitted, and aim to provide a model that has a more eloquent response to user interaction than the normal ‘grant’ or ‘deny’. The mechanism attempts to provide a framework within which the inhabitants themselves may develop appropriate access policies. With the vision of very large numbers of users inhabiting extensive virtual environments, the idea of ‘super user’, ‘group’ and ‘user’ seems too impoverished, and imposes an undesirable and aggressive hierarchy that is unlikely to have a meaningful correspondence in terms of the social layout of the space as it evolves. Thus we aim to provide a *collaborative* access model.

The access model

In this section we describe the high-level concepts of the collaborative access model. It is implemented using the Deva entity/component architecture. The model is based around the following main concepts:

- **Entity**: a general resource within the system. An entity may be graphically represented (such as a 'a house' or a 'a teapot') or may be an abstract concept with no visible representation (such as 'a group'). Entities contain an amount of 'state' and provide a number of 'methods' that alter that state (they are in most respects 'objects' in the 'object orientated' sense).
- **User**: a particular kind of entity that is related to a person in the real world
- **Key**: an entity that describes a user's relationship with a particular method of a named entity

Traditional object orientated languages/systems (e.g. C++, Java, SmallTalk, Eiffel) provide for a degree of access control between classes of objects. Thus, in C++ for example, it is possible to declare at compile time the methods and state of a class that maybe accessed by 'objects of the same class', 'objects of a specific class' and 'any object'. In C++ this is a rather crude mechanism based on 3 levels of privacy ('private to me only', 'private to me and anything derived from me', and 'available to all') with finer grained access made possible by declaring exceptions to these rules. The other languages provide finer grain and perhaps more elegant solutions to this problem. However all deal with access at what is essentially a compile-time level[1] and only have a notion of 'class' access rather than 'ownership' access. Thus it is either valid, or a compile-time error for Class A to attempt to access method B of Class C.

In our model, all entities communicate/interact by calling methods on one another. The access model is driven through this mechanism. That is, at run-time, whenever an entity attempts to perform an operation on another entity, the basic mechanism that makes the interaction possible intervenes and validates that operation before allowing it to continue. This is akin to every function call in C being intercepted and validated by some mechanism. The entities themselves are coarse grain enough, and the implementation of the access model lightweight enough for this not to be a performance issue.

The initial validation procedure is this:

1. *User A* attempts to call *M* a method on a second *Entity B*.
2. *Entity B* checks with the world's repository of 'keys' to see if a valid key exists for *User A* that would allow that user to successfully call *Method M*. If such a key exists and is currently valid (see later), then the call of *Method M* succeeds, and the operation continues.
3. If no such key exists then the operation is passed to Entity B as an 'authorise' query. That is, the system itself requests of Entity B, 'would you like to allow this operation to continue?'

[1] For Java and Smalltalk, the distinction between 'compile time' and 'run time' is somewhat blurred by their execution on virtual machines; nevertheless the intention of the languages are clear on this point.

4. Entity B is then free to either:
    a) grant the operation immediately in which case the result is identical to the outcome of step 2 succeeding (i.e. a key existing) or
    b) take further action related to that operation

Although on the face of it a straightforward 'grant/deny' strategy, a significant amount of flexibility is hidden within the above mechanism. The vast majority of the detail of the access model is contained within the idea of a 'key' and in Step 4b above. This is described in the following sections.

Detailed anatomy of a Key

A Key entity consists of three parts: a signature, a state and a payload.

- **The 'signature'** determines what method the key is able to 'unlock'; it contains the name of the user to which the key is attributed, the class of the entity on which it may be used, and the name of the method that it will activate and a unique 'generation' count.
- **The 'state'** is one of 'valid', 'invalid', 'pending' or 'unknown', and a key must have a 'valid' state in order to provide access to the method nominated in its signature.
- **The 'payload'** is an arbitrary piece of code that determines how a key's state changes when it is used (in reality, the 'piece of code' is actually a further method call made upon an arbitrary entity, however for now we will avoid what might seem initially like a meta-regressive situation by ignoring this issue!)

Keys are held in a 'global' repository called a KeyManager, which is a regular entity that responds to queries from other entities about the existence of nominated keys (again, we ignore the looming recursive complications for the time being).

The existence of a valid Key is enough to grant access to the nominated User/Entity/Method in its signature… this facility is provided by 'the system itself'[1] in dialogue with the KeyManager. The Keys too are real entities, and respond to an 'activate' method that may be called only by the User named in their signature or by the KeyManager itself. The consequences of activating a Key are to execute the code contained in the payload, the primary result of which is twofold:

1. The 'state' value of the key is changed to 'pending' until the activated payload code completes. Whilst in a pending state, keys do not respond to further 'activate' calls.
2. When the payload completes, it returns

[1] More dangerous recursion, since there really is not much of 'a system' to speak of; everything but the very low level communications is mediated by first class Entities. But these are details for later sections.

a) a new ‘state’ value for that key, and
b) optionally a replacement payload (if no payload is returned, the previous one persists)

As well as explicit activation by the User, a Key is activated by the KeyManager whenever an operation that could potentially result in the Key becoming valid is requested by an Entity. That is, when an entity queries a KeyManager, if a key with a matching signature is found and the key’s status is not ‘pending’, that key has its payload activated. If the status of the Key *prior* to the payload being activated was valid, the KeyManager returns a value indicating that a valid key has been found, otherwise a result is returned indicating either that no appropriate key exists, or that such a key exists but is currently invalid. Whatever the state of a key, however, as long as it has a payload, its state may change from invocation to invocation. The implication of a Key without a payload is that its state is fixed forever.

In this model, then, the payload can be seen as describing a set of criteria that if satisfied will make an operation possible. The criteria may be as simple and immediately deterministic as ‘you must be called Tom’, or ‘it must be Thursday’, or more complex such as ‘you must be a member of Group X’ or ‘you must have visited my homepage at least twice’ or even ‘you must have given me 10 credits’. The important thing is that the criteria are defined using the same techniques as normal entity method calls, and therefore have ‘access’ to any appropriate factors within the environment with which to make their decision as to whether to return a valid or invalid state. The payload may return ‘immediately’ for simple deterministic criteria, or may take an arbitrary amount of time polling other users, waiting for events to occur, before making a decision. In any event, the programming model is multi-threaded and asynchronous, and such delays only (correctly and by definition) hold up the call to the initiating method, allowing the user and all other entities in the system to continue their actions as normal.

This concludes the description of the basic mechanism that underlies the access model. An important point to note at this stage is that the mechanism is not provided by the VR system itself (as systems architects we consider that ‘hardwiring’ such a mechanism into the fundamentals of the VR system would inevitably lead to clashes with other requirements at a later date). Rather it represents a (somewhat advanced) use of the more general facilities provided by Deva for including detailed functionality on a per-environment (in this case The PLACE) level. All the functionality of the access model described thus far is written using the ‘component’ architecture and its notion of ‘imbued’, ‘innate’ and ‘enforced’ properties; it requires no ‘hacking’ of the core system.

Left in this state, the access model provides only low-level support for controlling user/entity interactions. In the following sections we examine the next level of entity functionality required to move the access model towards being realistically available to the inhabitants of the VE.

What has been provided so far is a basic mechanism enabling or disabling access to specified methods according to the existence or otherwise of 'Key' entities. Ownership of a valid Key gives access to a method; lack of ownership allows the entity to either refuse access outright, or more importantly to initiate a sequence of events that may eventually result in a valid key being generated. In the following sections, we examine the construction of entities that provide a vocabulary which enables a meaningful 'sequence of events' to be described.

UserRequest

The UserRequest entity interacts with a live user to solicit a response generated from the criteria of a Key. The essential functionality of a UserRequest is to return to the caller the 'status/payload' component of a Key. This is usually achieved through some form of user interface which allows the user to return simple 'yes'/'no' values through button presses, to select a more sophisticated pre-packed response from a list, or to code an entirely new response. RequestManager components of a user's client may provide such functions as a UserRequest bank (to allow asynchronous or offline response to UserRequests) or 'procmail' type filtering of UserRequests (e.g. 'always deny requests from MrAnnoying').

PollBooth

PollBooth entities are created by other entities when a response needs to be solicited from a number of other users before the decision as to what kind of payload to return for key can be made (a concrete example of the use of a PollBooth is given shortly). A PollBooth is created with a list of users, and textual string describing the purpose of the poll it is to carry out to the users. The PollBooth then sends a user-request to each of the nominated users, and then waits for a response from each of them. Immediately after having sent out all the requests, a PollBooth is able to respond to queries from the initiator of the poll concerning the number of valid (or otherwise) keys that have been returned. This allows the initiating entity to make decisions based on the progress of the poll; a 'consensus' decision might have to wait for all keys to be returned, whereas a 'two-thirds majority' poll might be able to complete if enough active keys are returned, regardless of the values of the outstanding keys.

Note that the responses from users are not simply 'yes' or 'no', but first class keys. When the initiating entity requests the status of a Poll, each returned key is activated, potentially changing the result of the poll on the next status request. This allows polled users to return complex criteria in the normal manner.

Group

A Group entity represents a collection of users. Variations on the theme provide different modes of membership.

- **BasicGroup**. These may be used to define the criteria for granting keys. All members of a BasicGroup have equal status, and joining and leaving a BasicGroup has no particular requirements and is simply achieved by requesting a change in membership from the BasicGroup entity. A number of variants of the BasicGroup provide more useful functionality in terms of managing access to other entities (note again these are not 'system' functions, but are essentially 'user defined entities').
- **OwnedGroup**. An OwnedGroup inherits the properties of a BasicGroup, but in addition nominates a single user that is the group's owner. Once created, the group's owner is responsible for granting or otherwise requests for membership (note that the key mechanism may be used by the group's owner to provide such notions as delegation of duty or automatic granting and denying of requests… an example of this is given later).
- **ManagedGroup**. A ManagedGroup contains two BasicGroups, one that includes regular users, and another that includes users able to grant and deny membership.
- **ConcensusGroup**. A ConcensusGroup contains one BasicGroup but moderates access to membership through requests to all existing members that must be validated before a new member is added.
- **AnarchosyndicalistCommuneGroup**. Similar to a ConcensusGroup except that power is derived from a mandate from the masses, *not from some moisten bint lobbing scimitars around*.

Although only a small subset of the possible variations of group management and access policies, the various operations provided by the Group, UserRequest and PollBooth entities cover many of the basic and common operations. A significant amount of tailoring of this functionality is possible using the Key mechanism, and (albeit advanced) users are free to create further variations of the Group entities, or indeed completely new forms of access object.

Presenting the model to users

The previous section presented a flexible and dynamic access model. The model is extremely powerful in that it allows a wide range of behaviors to be encoded. The use of keys provides an expressive mechanism that can be configured to support a wide range of activities. However, the flexible nature of the model means that presenting it to users represents a real challenge.

The core model allows a wide range of behaviors to be coded by application developers and to allow these to be amended if you have sufficient skills. In order

for general users of the environment to understand the access control in operation within the environment and to dynamically modify the model we need to consider the development of a presentation model. The presentation model builds upon the flexible core model and exploits the basic mechanisms to present an initial set of policy statements of utility to the model. The aim here is to allow users to understand their relationship to the environment in general. The presentation model does this by identifying and encoding a subset of user types and an initial set of object operations and a simple vocabulary for encoding the relation between users and entities.

The presentation model focuses on encoding access based on the entities making up the virtual environment. We use the term entity here to represent two main classes of object.

Zones: A zone distinguishes an area of the environment. Zones allow us to represent and control access to different subspaces. For example, in the mixing bay study we would say the mixing bay is a zone.

Object. Objects are used to represent entities placed within the space. For example, we would represent a video player as an object and the room it is placed within as a zone.

The presentation model handles Zones and Object in an orthogonal manner with the same set of user relationships being applicable to both Zones and Objects.

Representing users relation to the environment

The presentation model identifies a range of different types of user. For any object a user will be seen as one of these different user types. Identifying these different user types allows us to make general statements about access to models and the relationship between an object and an as yet to be identified class of objects.

The model identifies the following relationships between any entity within the environment and potential users:

Owner: An object owner (or owners) has ultimate responsibility for the entity and is normally the person who created the object or where given the object by another user. Often owners have little day to day involvement with the object. For example, the mixing bay reported in chapter 2 is owned by Lancaster University .

Manager: An entity manager is a person (or group of people) who have day to day control of the object. For example, in the case of the mixing bay study (chapter 2) we say that three members of support staff managed the notice board.

Inhabitant: Entities exist within some space and that space is inhabited by a number of well-known users. The presentation model allows you to distinguish between these users and more general users to offer a richer set of behaviours.

Visitor: This is the most general relationship available and is used to handle the broadest set of behaviours for people who are recognised as visitors to the environment.

Everyone: This relationship is used as a catch all to handle default behaviour.

All of these relationships can be parameterized with the name of a particular zone allowing the model to specify specific inhabitants and to distinguish differences between them. For example, we can represent all inhabitants of the computing department as **inhabitant***(computing_department)*.

Representing permissions in the environment

The presentation model also provides an initial set of key permissions that are used to represent access within the environment. These are the basic set of operations used to encode access control. The presentation model provides these operations in terms of a Zone centric perspective. Thus permissions are expressed mainly in terms of Zones

Zone Permissions

The Zone permissions provided by the basic presentation model focus on controlling access to and the amendment of different spaces in the environment. The basic operations are

Enter This permission allows an identified class of users to enter the zone.

Leave This permission allows an identified class of users to leave the zone.

Take This permission allows an identified class of users to take an entity from the zone. This may be parameterised to identify particular entities.

Put This permission allows an identified class of users to place an entity within the zone. This may be parameterised to identify particular entities.

Add This permission allows an identified set of users to create and add objects to the zone. This may be parameterised to identify particular entities

Delete This permission allows an identified set of users to destroy entities within the zone. This may be parameterised to identify particular entities

This set of simple zone operations focus on managing the space by controlling both access to the space and the addition and removal of entities within the space. The model is complemented by a set of entity permissions.

Entity Permissions

Entity permissions within the basic model identify the core means of controlling the use of the entities within the space. The basis of the model is that for each

method provided by the entity we may provide permission to use that method. Essentially two permissions are provided:

See (methodname): This permission allows an identified class of users to see a method provided by an entity. When no parameter is given the permission applies across the whole object.

Use (methodname): This permission allows an identified class of users to involke a method provided by an entity. When no parameter is given the permission applies across the whole object.

Describing permissions in the environment

The use of zones and entities and the permissions identified above allow users to encode different forms of access within the environment. To further support users we provide a set of access tables that can be completed for each zone and object.

The Zone access table

The basic zones access table encodes information in terms of the basic permission outline in the previous section. The general structure is of the form:

Enter

User Relation	Action

Leave

Object	Action

Take

User Relation	Object	Action

Put

User Relation	Object	Action

Add

User Relation	Object	Action

Delete

User Relation	Object	Action

You will notice that each entity in the table provides an action field. The addition of these action fields allows users to be notified about the access to the environment. Two main actions are provided as default:

Inform This tells identified users that access has occurred

Request This seeks a response from users before an access is granted.

These two actions map down to the basic mechanisms provided in the core access model. The use of the access table is best identified using an example. Consider an public space where people can move entities if they live there but if

they wish to remove an object the object owner must be asked. The access table associated with that zone would be of the form:

Enter

User Relation	Action
Everyone	

Leave

Object	Action
Everyone	

Take

User Relation	Object	Action
Inhabitant	*All*	*Request.Manager*

Put

User Relation	Object	Action
Inhabitant	*All*	*Request.Manager*

Add

User Relation	Object	Action
Inhabitant	*All*	*Request.Manager*

Delete

User Relation	Object	Action
Inhabitant	*All*	*Request.Object.Owner*

Note the use of the notion "Object.Owner" to refer to the appropriate relation drawn from the set of objects identified in the object parameters. The access table can also be used restrict access consider a personal space where only the manager of a zone can enter the space and exploit the objects in the space but the owner of the space is the only person who can modify the space.

Enter

User Relation	Action
Manager	

Leave

Object	Action
Manager	

Take

User Relation	Object	Action
Owner	*All*	

Put

User Relation	Object	Action
Owner	*All*	

Add

User Relation	Object	Action
Owner	*All*	

Delete

User Relation	Object	Action
Owner	*All*	*Request.Object.Owner*

The use of the presentation model allows us to represent a rich set of relationships between the zones making up a space and the consequential behaviours. The access tables associated with zones and entities also allow us to consider the development of a simple form-filling interface to allow users to configure the space. Each zone and entity in the space would have a

corresponding access table that can be called up and then filled in reflecting the different permissions associated with the zone or entity.

Part II: The Abstract Electronic Landscape

Chapter 6: The Abstract eSCAPE Revisited

John Mariani and Andy Crabtree
Lancaster University

Last year's work saw the design, implementation, and end-user evaluation of the *Library Abstract eSCAPE Demonstrator*.[1] Following the situated evaluation of the library demonstrator,[2] an effort was made to identify a set of technical issues that would inform the next iteration of the demonstrator and which could be addressed by users in the next evaluation of the system. The purpose of this chapter is to review the issues that were raised by users and to list the potential solutions applied to the next generation of the system.

The issues: descriptions and solutions

Here we present each issue identified by the users and, in most cases, potential solutions. The issues are not presented in any particular order and range across a wide spectrum. Some concern relatively trivial problems that have an easy technical fix, some have deeper technical impact, whereas others relate to users' perception of the fundamental concepts underpinning the system.

Reliability

Although sufficient for demonstration purposes, at a practical level the demonstrator at the i3net conference in Sienna (hereafter referred to as the "Sienna Demonstrator") 'crashed' easily and often. While the Q-pit space generator is a fairly complex piece of software, it was our belief that the cause of the crashes was the DIVE visualiser. It should be borne in mind that Q-pit is one of the first DIVE applications to have a predominantly dynamic nature—most DIVE worlds are statically defined and loaded into the visualiser. By way of contrast, Q-pit has to introduce each book dynamically into the space and the links and clouds also have to be added. One observable consequence was a reoccurring 'timeout' problem whereby DIVE would fail to create an object in the first instance but then attempt to manipulate the object (during the settling-down period of the force displacement activity, for example) and hang the system

---

[1] eSCAPE Deliverable 4.1 (1999) *The Library Abstract eSCAPE Demonstrator* (eds. Mariani, J. & Rodden, T.), Esprit Long Term Research Project 25377, ISBN 1 86220 079 3.

[2] Crabtree, A. (1999) *Situated Evaluation of the Demonstrator*, The Library Abstract eSCAPE Demonstrator (eds. Mariani, J. & Rodden, T.), eSCAPE Deliverable 4.1, p. 109-146, Esprit Long Term Research Project 25377, ISBN 1 86220 079 3.

waiting for a timeout. Consequently, the system would crash. This situation, which was mainly occasioned when large searches were undertaken, was handled during the situation evaluation by using the standard WebOPAC to identify manageable searches prior to issuing the search via the demonstrator. On other occasions when crashes occurred, the designer doing the demonstration would restart the system as quickly as possible and re-establish the search space being explored immediately prior to the crash. The situation was such that it would be surprising had the users not commented on the unreliable nature of the system. Potential solutions to the problem consist of:

- Modifying the DIVE software so that it handles the dynamic nature of Q-pit visualisation more reliably.
- Porting the Q-pit software to a different visualiser.

Both avenues of solution have been explored in the course of this year's work and porting has proved to be more effective within the time constraints.

Scalability

For a number of reasons, the number of books that could be returned in a Q-space was limited to around seventy-five. This was quickly highlighted as a problem by the users, who expected the space to be much larger returning hundreds (if not thousands) of 'objects' (i.e. books, journals, videos, etc.). On a technical level, seventy-five objects seemed rather low for 'proving' the worth of the visualisation technique. However, the number of objects handled was kept low as the calculation of the Q-space involves an order N-squared operation. The sequence of calculations runs as follows.

- Similarity matrix
- Minimal spanning tree
- Region identification
- Force displacement

The similarity matrix calculation is of order N-squared. More specifically it involves calculating all the values for the strict upper triangle of a square size n matrix, which for n objects gives you $n*(n-1)/2$ similarity computations. To complicate matters further, some of the similarity comparisons themselves contain N-squared functions. Similarity comparison between two floats or a string is fair enough, but if we are doing word-by-word similarity within titles, that too is roughly m-squared according to the number of words in the sentence. The other operations involved the following magnitudes:

Operation	Magnitude
Minimal spanning tree	Order *n*, with the 'depth' of the tree Order(*n**log *n*). The test is not free but it is fairly cheap
Region identification	*n**log *n* to do a quicksort, followed by a linear traversal (*n*).
Force displacement	*n***n* + *n*Log *n*

Thus, increasing the number of objects increases the amount of time taken to generate the space. The impact of increasing the number of objects to hundreds and even thousands is two-fold:

1. There will be an immediate implication on the algorithms used to generate the Q-space. Current algorithms can only cope with small numbers of objects and so new variations of the algorithms have to be created.[1]
2. There will be an immediate impact on the visualiser chosen to implement the next generation of Q-pit.

Possible solutions to these problems were delayed until the actual implementation of the next generation, and are discussed in detail in the next chapter.

Full title not available

The original demonstrator presented the title of an object in three possible ways. If the labelling of the 3D objects in the space was set to ‘title’ then each object’s label represented the title of the object. The Head Up Display (HUD) also showed the object’s associate record, of which title is a part. Finally, when a book was selected, a pop-up window also contained the object’s record. In every case, however, the title of the book was often truncated. From an end-users point of view, the title of a book is one of the most (if not the most) important indicator of the potential usefulness of any particular object. The title was only partially available, however, and searching was seriously hampered as a result.

The technical reason for the truncated object title may be described as follows. The original demonstrator had a two-level database architecture. Closest to the OPAC library information was a MySQL relational database, capable of acting as a cache to keyword requests. The Q-pit system itself visualises the contents of a ‘local’ Q-pit database which simply holds a set of tuples to be visualised. The Q-pit database management system is a simple Java class, with fixed-sized fields. This is the source of the problem. The maximum size of a field is (say) 50 characters. Thus object titles greater than 50 characters were truncated. It should be noticed that both the OPAC and MySQL databases do not truncate this information that is therefore available. This problem then reduces to a

[1] One way to cope with larger numbers would be to reduce the exactness of matches and to generally loosen up on the placement of objects so that they are approximations rather than exact positionings.

straightforward technical fix: we need to substantially increase the maximum size of a field within the Q-pit database.

Inconsistency of colour

Colour was used in the original demonstrator to indicate the existence and membership of a region. The Q-pit undertakes the generation of regions of objects based on their similarity. Each region then has a colour associated with it. Objects within that region are given the same colour. Where a region has three or more objects they are encased in a hull (or cloud) of the same colour. To give a general example of the troubles experienced by users here, with a single visualisation a user may select an object within a region and discover it is a book. They may even click on a further regionally located object and it too may be a book. The user may then draw a conclusion that all objects within that region (or having the same colour, green say) are books. They may also conclude that all books in the space are green. The other clouds consist of other types of library objects - CD-ROMs, oversized pamphlets, videos etc.

The 'inconsistency' of colour arises because conclusions of 'similarity' may well transpire not to be so. Regional similarity is of course dictated by the weights that generated the space. Having been instructed in the 'same colour, similar type of object' principle of searching, however (that red objects are similar, green objects similar, and so on), users were somewhat surprised to find that members of particular colour classes were not always similar. When doing a 'publication type' search, for example, users found that green object's could be books, videos, CD-ROMS, over-size pamphlets, and the rest!

This is a rather difficult problem to deal with. One possible solution is to assist the user with their interpretation of a cloud, to try to avoid the drawing of a misleading conclusion. This could be achieved by processing the contents of a cloud or a region and generating a general summary of what is in the space. For example, part of the cloud description might include the fact that the cloud is '80% books' that indicates that the other 20% is spread through other types of library objects. This should help defuse the incorrect inference that all green objects are books, say.

Other aspects of colour use are harder to deal with however. If the user chooses to change the weightings and regenerate the space, for example, the work is undertaken as if it were a completely new search. Regions are newly generated and colours are again randomly allocated. This means that the new green region is potentially completely unrelated to the old green region. We can't take forward our knowledge of prior green things and relate them to the green things in the current space. One potential solution to this is to maintain information about previous spaces, to recall that these items were green in a previous incarnation. We would then examine the new regions and look for a majority of prior colour and repaint the region appropriately. The power of this possible solution is

doubtful, however, as we would always have objects that were not green before now being coloured green. Another potential solution is to emphasis the difference between the spaces. If we choose to use shape to represent 'iteration through this query space' then the first visualisation could consist of all cubes; the second of spheres, the third of pyramids and so on. This would hopefully emphasise that green cubes are not the same as green spheres. Yet another solution would be to broaden the colour palette used so that, as far as possible, colours are not repeated between visualisations, so that inconsistencies could not arise.

Individual items and classmark relations

Classmarks within Lancaster University's library classification systems are used to group similar objects close together on the shelves of the physical library. For example, programming books have a classmark of something like AZK. If by searching catalogues manually or online a user finds an object he or she is interested in, they can go to the appropriate shelf and find the object. In addition to finding the object, the user also places him or herself in a zone of the library whereby the object is surrounded by objects covering the same (or very similar) topic. Q-pit searches will also find objects of the same classmark, but not necessarily all the objects of the same classmark. For example, searching for a book on Java programming will find all books on this topic, but not books on Ada or Pascal or C programming, which the user may well be interested in. Some mechanism is (thus) needed whereby once having found an object of interest in the Q-space, it is possible to generate a new Q-space containing all other objects of the same classmark. It is technically possible on OPAC to do a classmark search, and so Q-pit needs to be extended in someway to

1. Allow the user to specify that they wish to see all objects of the same classmark of the current object (one way might be to let the user 'fly' into an object).
2. Allow Q-pit to do the classmark search and to generate the classmark Q-space just as it normally generates keyword spaces.
3. Allow the user to return to the original Q-space once they have completed their exploration of the classmark space.

Centred view

This is a relatively trivial issue that nonetheless needs to be addressed if the demonstrator is to be of practical utility. In the original demonstrator, the start position of the viewer was fixed. It may well have been that this position was off to one side of the Q-space generated. The user trails suggested that the start

position of the viewer needs to be altered so that the user is facing the centre of the space and the majority of the space should be within their view. This is a far more suitable starting place for subsequent browsing of the space and easily resolved.

Clicking on objects within a cloud

A technical visualiser problem that arose during the evaluation is that when clicking on an object contained within a cloud from outside the cloud, the hull is the object that is selected. This problem is, we believe, a visualiser concern that can be addressed.

Overlapping objects

Users frequently experienced difficulty in selecting an object when it is very close to another (overlapping) object. This is an issue that relates to the overall density of the space or to specific regions. Steps should be taken to decrease the density of the space, and to provide the user with appropriate navigational aids to move around the space so that the object they are concerned with is no longer occluded. The problem is, in many respects, an aesthetic one concerning the 'layout' of the visual field.

Related keywords

Users are becoming much more sophisticated as they are exposed to a great many search engines in their everyday computer-based work. As a result, they become used to facilities on other engines and expect them to be available. One commonly found feature is that of related searches. We even find this kind of information in commercial web sites such as Amazon (functions such as "readers who bought this book also purchased …". OPAC does not offer this feature. Nevertheless, the Sienna demonstrator (constructed after the original trials), adapted the *10_dencies* art piece to provide such information in presenting a visualisation of previous search 'trails'.[1] The notion could, in principle, be readily extended to generate a Q-space of related 'trails'.

[1] For further detail of the use of *10_dencies* and search 'trails' see eSCAPE Deliverable D4.3 (2000) Configuring Blue-Sky Research Concepts, Common Methodology, Esprit Long Term Research Project 25377

Difficulty over the concept of weights

Users appeared to have some problems with the idea of ‘weightings’. From a design point of view, this is a something of a ‘problem’ as it is one of the fundamental concepts of the whole system. One potential solution to this is to change the nomenclature of the concept. To what is an open question (although the readily intelligible categories of ‘search options’ or ‘search criteria’ have been suggested). For demonstration purposes, this has not been accorded a high priority however.

Supporting searching by elimination

It is possible when exploring the space to identify objects or even clouds of objects which the user considers not to be relevant to their search. A keyword search on Java, for example, will retrieve books on the programming language and on the island. Suitable weightings (for example, on classmark) make it possible to separate out these different classes of object into distinct clouds. In doing this, users suggested that it would then be useful to be able to say “this book or this cloud does not meet the kind of thing we are looking for, please delete it from the space”. This should remove objects from the space that would not participate in any further regenerations of that space. This would require Q-PIT to be able to:

1. Allow for the selection (and highlighting) of books and of clouds
2. Allow for the deletion of highlighted objects from the space

Relating separate searches

Users presented a need for the ability to be able to take books from prior searches and to combine them together into a new space. That is, to take as the source for the generation of a Q-space objects that already exist in the space, rather than to return to the OPAC or to the MySQL database. In combination with the ability to delete objects and clouds from the space, it should be seen that this provides a powerful mechanism for manipulating existing spaces and for finding out things about combinations of spaces that hitherto would not be possible to do. Relating searches would require Q-PIT to be able to:

1. Allow for the co-existence of Q-spaces within a single Q-pit. This should be possible by simply allowing users to ‘move away’ from the current Q-space into a clear area of the overall space and to generate a new Q-space at that location. Moving position in the overall space allows then allows users to revisit previous spaces.

2. Allow for the selection of clouds from multiple Q-spaces and make them the source for the generation of a new Q-space that combines the contents of highlighted clouds.

Because of technical considerations, the performance of visualisers degrades the more populated they become. This means we have to provide a mechanism for users to be able to delete entire Q-spaces as well as individual objects and clouds

Positioning of object text in space

Unlike the Sienna demonstrator, the version used in the situated evaluation did not have a Head Up Display (HUD). In the Sienna Demonstrator, the HUD is used as one of the options for displaying the record that describes the selected object or book. In the evaluation system, the record was displayed adjacent to the circular navigation cursor in the centre of the screen, and this was thought to be difficult to read. It might appear, for example, overlaid on an occluded object and, as such, not against a black background. Attempting to navigate away from the occlusion would also likely take the navigation cursor away from the object of interest and cause the overlaid information to vanish. Our attempt to solve this problem with the Sienna Demonstrator consisted of introducing the HUD which was placed at the top left hand corner of the overall display on a transparent blue background preventing occlusions. While discussing this issue, another point raised regarded the pop-up window with the book record contained within it: the basic issue was "why"? It was felt that this additional display was not required (or at least one of the two was not required).

Returning to an object

Another practical problem emerging from the user evaluation: once a user has identified a useful object, how can they return to it? We would expect users in their normal explorations to find books of interest but then to browse away from them, trying to find others. Moreover, even if the user tries to stay in the neighbourhood of an interesting book, if they choose to reorganise the space (by changing weights), how would they find the object again? Clearly, some 'bookmarking' mechanism is required here. A bookmarking operation should be available which causes the selected book to become highlighted in some manner. In earlier versions of Q-pit, an enclosing wire mesh was used for this kind of highlighting. This highlighting should be persistent within a Q-space and survive regeneration of the space. A further mechanism that would be useful here is that of a 'history list'. The user's movement through the space should be recorded and 'forward' and 'backward' buttons should be provided which allow the users to step backwards and forwards through their selections. Again, this history list should survive regenerations of the space.

Summary

The purpose of this chapter has been to revisit and *highlight* the practical issues arising from the use of the library demonstrator in the course of its situated evaluation by end-users as reported in Deliverable 4.1. We have tried to describe the issues of concern as identified by users and to briefly outline potential technical solutions to the 'problems' encountered by users in confronting the library demonstrator with practical circumstances of actual use. Those circumstances brought the following issues to bear on design and have shaped this year's work to some, not inconsiderable, extent:

1. Constructing a reliable demonstrator that does not 'crash' when confronted with real world tasks.
2. Scaling the demonstrator up to handle large searches (+350 objects).
3. Furnishing users with the full title of objects so that may conduct real world searches for information.
4. Reducing inconsistencies in colour usage.
5. Supporting topic-based searching by enabling users to browse objects that share the same classmark from any point in the course of searching results.
6. Supporting navigation in 3D space in providing a 'centred' view.
7. Providing more flexible means of navigation to allow the more effective manipulation of specific objects located within data clouds.
8. Supporting navigation in providing an efficient aesthetic layout of the visual field.
9. Developing 'related keyword' search functions.
10. Clarifying basic conceptual terminology reflected in the user interface.
11. Enabling searching to proceed by a process of elimination.
12. Enabling multiple searches to be related and queried.
13. Providing more legible or readable object text displays.
14. Providing a history function so that users may retrace the steps taken during any particular search.

Work on the library demonstrator has been directed towards addressing as many of these issues as practically possible. In the following chapter we report on the specific solutions implemented in response to the outcomes of the situated evaluation prior to reporting on user experiences with new library demonstrator.

Chapter 7: The Library Front end -Technical Realisation

Steve Pettifer, John Mariani
The University of Manchester, Lancaster University

Virtual Environments have attracted attention as locations for abstract data visualisation since their inception; as well as providing a ‘third dimension’ for locating data, their analogy to our real world has lead to speculation that natural human perceptual mechanisms may come into play, bridging the gap between comprehending complex data and the limitations of the algorithms for arranging it in space. Many demonstrations of populated information terrains have been implemented. However, as with many other fields in virtual reality, by far the majority of these have either been three dimensional environments so complex as to be unable to operate in under the rigorous requirements of virtual realty, or have been essentially ‘passive walkthroughs’ of pre-computed data. Both these present limited opportunity for virtual reality's novel properties of presence, ‘direct interaction’ and ‘spatial awareness’ to be exploited.

Some of the impasse in data visualisation in virtual environments can be put down to limitations of the hardware of computation and of graphical rendering. However, both these fields have demonstrated dramatic improvements over recent years... improvements that are not on the whole mirrored by the VE implementations of data visualisation.

In this chapter we present the 2nd generation implementation of the Q-PIT, a general-purpose browsing tool for locating objects of interest with an 'abstract electronic landscape'. We focus on issues of 'scalability' of such a tool for virtual environments arising from a previous implementation and its study in-situe, consider the architectural implications of its construction, and present novel algorithms for optimised rendering, structuring and interaction with the environment. Finally we present an architecture suitable for making the Q-PIT into a multi-user environment.

Aims

The aim of the Q-PIT work has been to explore appropriate technologies and techniques for building large scale electronic landscapes-that is to say, multi-user virtual environments that have social affordances as well as being sources of information. The Q-PIT represents the 'abstract' end of a spectrum of such environments to which a cityscape-like environment forms the 'concrete' end.

A particular aim of the work was to establish what the social affordances of abstract spaces might be. This implied having an abstract space that was of an appropriate scale to provide a realistic task for a group of users to engage in, and also to provide enough appropriate functionality within the environment to keep users engaged (i.e. there must be 'something real to do' in the environment; a problem that often dogs the builders of VEs).

Initial studies of users of a library[1] provided a basis for the design of the first generation of Q-PIT, and guided the particular functions that were implemented.

The Environment

The Q-PIT is a populated information terrain[2] consisting of objects positioned within a three dimensional space, connected by lines representing relationships, and grouped within convex ‘hulls’ representing similarity. In principle, the objects can represent any data items from a database, since once a similarity function has been defined for the particular objects, the remainder of the generation of the Q-PIT is based on similarity rather that on the contents of the particular data items.

1 A. Crabtree, "Understanding searching as a socially organised achievement in the library" (Chapter 1), Escape Deliverable 4.1. (1999) Esprit Long Term Research Project 25377, ISBN 1 86220 079 3

2 J. Mariani, J. Trevor, M. Stenius, and D. Schwabe, "The abstract electronic landscape demonstrator" (Chapter 5), Escape Deliverable 4.1. (1999) Esprit Long Term Research Project 25377, ISBN 1 86220 079 3

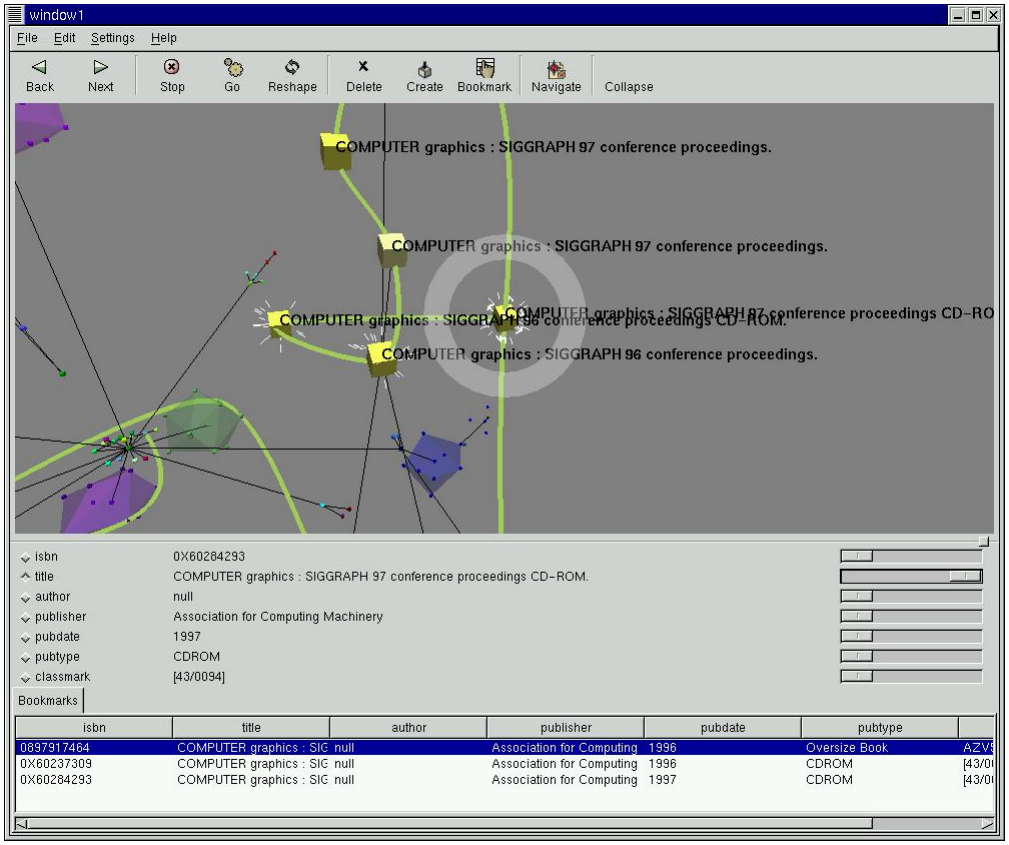

window1
File Edit Settings Help
Back Next Stop Go Reshape Delete Create Bookmark Navigate Collapse
COMPUTER graphics : SIGGRAPH 97 conference proceedings.
COMPUTER graphics : SIGGRAPH 97 conference proceedings.
COMPUTER graphics : SIGGRAPH 96 conference proceedings.
isbn 0X60284293
title COMPUTER graphics : SIGGRAPH 97 conference proceedings CD-ROM.
author null
publisher Association for Computing Machinery
pubdate 1997
pubtype CDROM
classmark [43/0094]
Bookmarks
isbn title author publisher pubdate pubtype
0897917464 COMPUTER graphics : SIG null Association for Computing 1996 Oversize Book
0X60237309 COMPUTER graphics : SIG null Association for Computing 1996 CDROM
0X60284293 COMPUTER graphics : SIG null Association for Computing 1997 CDROM

Figure 1: The re-implemented q-pit browser, showing the graphical user interface and a snapshot from a user's session

Our first implementation of the Q-PIT idea used the DIVE Virtual Reality system from the Swedish Institute of Computer Science as a visualiser. DIVE provides a shared virtual environment that can be populated with geometric objects, with their behaviour generally governed by scripts written in Tcl. The computationally intensive nature of the calculations that form the Q-PIT made writing such code in Tcl unrealistic; instead a separate process written in Java calculated the positions and groupings of the objects, and communicated with the DIVE VR component using 'JIVE', the 'Java/DIVE' bindings. Though JIVE provides a significantly richer programming environment than does the Tcl binding, it nevertheless presented problems of scale.

The significant delay involved in computing the placement of the objects required a decoupling of the algorithmic component from the VR system itself, and a series of progress bars and intermediate results were presented to the users of the environment, rather than waiting until the whole solution was achieved before displaying anything.

Using Tcl to implement object behaviour within the DIVE engine limited the amount of computation possible within the VE itself, both in terms of performance and in terms of the complexity of programs that are realistic to write in Tcl. This meant that the addition of hulls and connecting lines to the

environment had to be calculated outside the VR system, and loaded separate objects after the solution had converged enough for hulls to be visible.

Architectural bottlenecks

Our initial vision for the Q-PIT was a very ‘dynamic’ space. Although the individual objects that form its contents are individually geometrically simple, in large numbers they represent complex entities as embodied by the hulls. Thus the relationship between an object and its position in space is non-trivial, and depends on the current wishes of the user (in giving an initial keyword and modifying the weights) and the behaviour of the algorithms. The primary architectural difficulty encountered in our initial implementation of the Q-pit was the division of the system into ‘algorithmic space’ (as embodied by the tuple space, similarity matrix, spanning tree and so on, and the algorithms that operate upon these structures) and the three dimensional space provided by the VR system. Transforming the results of algorithmic space into instructions that can modify the 3D space, and communicating these to the VR system proved an unrealistic task to be done in real-time. Communicating in reverse across this divide (i.e. relating non-trivial operation on the objects in 3D space back into manipulations of the tuple space or similarity matrix) proved even more complex.

A solution that we have found effective in other situations[1] is to remove the distinction between the VR system's internal representation of the space and the appropriate ‘application’ or ‘algorithmic’ structures appropriate for performing optimised computation of behaviour or dynamics. The MAVERIK/Deva VR systems both embody this approach, and the re-implementation of the Q-PIT using these tools is described later in this chapter.

The Q-PIT process revisited

The process of generating a Q-space is outlined in last year’s deliverable[2]. We revisit it here briefly and then consider improvements to each of the various stages.

---

1 R. Hubbold, J. Cook, M. Keates, S. Gibson, A. Murta, A. West, T. Howard and S. Pettifer, MAVERIK: a micro-kernel for large-scale virtual environments", in Proceedings of ACM VRST99, Dec 1999, London, UK

2 J. Mariani, and A. Colebourne, "Q-PIT and Dataclouds : the generative algorithms" (Chapter 3), Escape Deliverable 5.1. (1999) Esprit Long Term Research Project 25377, ISBN 1 86220 081 5

- The process begins by establishing a tuple-space formed by extracting on a keyword search basis whatever information is required from a larger database. In our implementation we have worked with 'books' and 'files'; the former being the object of the user studies and being retrieved from a 'live' library database, and the latter being a lightweight means of testing the Q-PIT software without the need of a connection to the 'outside world'. The objects in the tuple-space must have a comparison function defined on them that for each attribute can return a value between 0 (not similar at all) and 1 (identical) when compared with the corresponding attribute of a different tuple of the same schema.
- The next stage is to generate a similarity matrix for all objects; this involves a comparison of every object with every other, populating the matrix with a similarity value generated by combining the comparison functions over all attributes, weighted according to user-defined weights. This yields a triangle of a matrix with overall similarity values for each pair of objects.
- An ordered list is then generated from the matrix, positioning the most similar objects at its head and the least similar at its tail.
- A minimal spanning tree is then generated from this list (see previous deliverables for the algorithm)
- The tree is then traversed, 'painting' each of the nodes a particular 'colour' to indicate which grouping that node belongs to. A node is considered as belonging to a different group to its parent if its similarity value to that parent is less than a given threshold. A large threshold yields a small number of highly populated groups; a small threshold splits the tree up into a larger number of sparsely populated groups.

Algorithmic bottlenecks

Both our initial aspirations for the Q-PIT and the expectations of real users from studies of the initial implementation include the visualisation of a large number of data objects using the technique. In order for the technique to be a practical means of investigating data spaces, we must interpret 'large' in this sense as at least 'significantly more objects than could be dealt with using a list'. Furthermore, for the system to be of practical utility, its response time must be within that of a typical web search or library query. We set our goal for the re-implementation to be the formation of a navigable space of around 2000 objects in the order of ten seconds.

All three phases of the Q-PIT forming algorithms are computationally intensive. They are considered in detail in the following sections.

Similarity computation

The formation of a basic similarity matrix involves the comparison of every object in the tuple space with every other object. For *N* objects this populates a strict lower (or upper) triangle of an *N***N* matrix, giving *N**(*N*-1)/2 comparisons. Comparisons between the different types of attribute value have different computational costs:

The comparison of two numbers, whether floating point or integer involves establishing a range of values for that attribute, and then determining the ratio of the distance between the particular values with respect to the size of the domain. E.g. given a max and a min, the similarity of two values is 1-(|a-b|/max-min). This is a comparatively lightweight computation.

A number of different approaches can be taken for determining the similarity of a pair of strings. The simplest approach, which is suitable for a number of different semantic classes of strings (e.g. names, filenames) is a simple lexicographic comparison. This is linearly complex according to the number of characters being compared. A more sophisticated variation involves comparing the number of common characters in the string, perhaps weighted according to their position. This is typically of *N***N* complexity. More sophisticated still is to count the number of similar words within a string. This is also of order *N***N*.

Interacting with the database

Behind the visualisation represented by the Q-PIT lies a database; in our case it is the Lancaster University OPAC system, filtered through a number of layers. Though there are many tempting technologies for accessing such databases (e.g. various Object Request Brokers, CORBA, and mechanisms for 'directly' linking to the data structures of the database such as JDBC), it has been our experience that such conduits do not lend themselves well to direct interaction with a virtual reality system. The 'real-time' constraint of virtual reality means that often the data structures within the database must be traversed in very particular ways at very high speeds. Though it is possible to build indexes according to particular criteria in most database systems to optimise access, the type of dynamic traversal required for the real-time rendering of virtual environments is not possible. Thus it was necessary to write conduits that allow sections of the database identified by user defined keywords to be downloaded into an optimised 'tuple space' which is more amenable to manipulation by the particular algorithms required to build the Q-PIT.

The abstract class for a tuple needs to provide the following interface:

- Make me
- How many attributes do I have?
- Compare attribute *N* from myself with the same attribute of another tuple
- Get label for attribute *N*
- Get value for attribute *N*
- Return my 'shape'
- Return my 'size'
- Delete me

The 'shape' and 'size' provide an arbitrary mapping from the object to the visual representation. For a file, these are obvious. The colour of an object is determined by its position in the hull.

The identification of this abstract class allows us to also address the general issues involved in computing similarities. This has allowed us to develop an architecture that provides a generic Q-PIT server. This server provides an abstraction across a range of information sources. These include the University OPAC system as well as Z39.50 and information sources such as Amazon. The general server isolates the core algorithms for developing the similarity matrixes and clusters and then presents these to a range of interfaces. This arrangement allows a heterogeneous set of interfaces. The server is described in more detail in Chapter 10 with the different interfaces enabled by this generic architecture described in Chapter 11 and the discussion of alternative interfaces.

Force placement algorithm

The Fruchterman and Reingold force directed placement algorithm used in the previous DIVE implementation of the Q-PIT is an adaptation of the spring-embedder model, that follows generally accepted aesthetic criteria for graph drawing. This means that the final position of the objects pushed apart by the simulated forces have evenly distributed vertices with minimised edge crossing and controllable (or uniform) edge lengths. The algorithm works by calculating repelling forces for all participating objects, and then calculating attractive forces for only those nodes connected by edges. According to Fruchterman and Reingold, nodes at distance *d* are attracted to one another by the force f_a:

$$f_a(d) = d^2/k$$

and they are repelled (if connected by a node) with the force:

$$f_b(d) = -d^2/k$$

where *k* is the optimal distance between nodes in the graph, determined by the magnitude of the graph and the size of the final space that is to be generated.

The algorithm is iterative. Within each iteration the attractive and repulsive forces are calculated for each node, and then at the end of the iteration all nodes are given their new position before the next iteration occurs. A 'cooling' function is applied to the forces generated such that the algorithm is guaranteed to 'settle' into a solution (since the algorithm does not use balanced forces, without such a cooling factor there is little chance of a stable situation ever being reached).

Calculating the final force placement layout of the space is clearly a computationally intensive task. For the thousands of objects we are aiming at, such computation can not be completed within the $1/30^{th}$ of a second 'realtime' constraint of the virtual environment. For this reason, and for one of aesthetics, it is important to be able to show intermediate results from the force-placement routine. Conveniently, the iterative nature of the algorithm as it generates increasingly good placements is ideal for our needs.

Though the algorithm provides aesthetically pleasing results for almost any configuration of nodes, and provides easy access to intermediate results, this generality is bought at a significant performance cost. The algorithm is essentially of the order $N*N$ since one of the two sets of forces is calculated between all possible pairs of objects. Thus calculation of the repulsive forces dominates the processing time for all but the most densely linked graphs. Having re-written the algorithm in optimised C, with heavy use of pointer dereferencing to minimise the number of memory copies being performed during each iteration of the algorithm, it became clear that for any significant number of objects (our goal being around 2000 objects) the algorithm was far from being able to generate the interactive frame rates necessary (including the rendering times, an initial run of the C version of the algorithm for 2000 objects took nearly seven hours to settle into a compelling layout… it is easy to forget how quickly an N*N algorithm becomes unusable!).

Our solution to this problem was to exploit the hierarchical nature of the graph generated by the minimal spanning tree stage of the Q-pit's generation. Though Fruchterman and Rheingold algorithm is capable of dealing with arbitrary connected graphs, including cycles, the structures we generate from previous stages are always tree-like (that is, they are directed and acyclic). More specifically, they contain a hierarchy of distinguishable 'groups' (that is, after all, what the previous stages of the Q-pit algorithms have tried to produce). By the time we come to the force-placement, the tree structure has been 'painted different colours' to identify each group. The underlying structure is still strictly a tree at node level, but in addition to this represents a coarser grained tree at the group level. We take advantage of this layout in our new placement algorithm.

The new placement algorithm, which only displays benefits over the Fruchtermann and Rheingold version for hierarchical data, operates thus:

- Considering only the set of dominant nodes, use the original F&R algorithm to position the dominant node of each group in space.

- Considering then in turn each group represented by a dominant node, perform the F&R algorithm on all nodes in that group, and position them in space relative to the current position of their dominant node.

This relatively small modification to the F&R algorithm reduces the overall complexity from N*N to being Nlog(N) which for large N is a significant improvement. The disadvantage of the algorithm is a small loss in terms of the aesthetics of the layout, since what is being computed is now a number of individually optimal layouts within a larger layout, rather than a globally optimal solution. In practice however, the difference is negligible as shown in figures 1 and 2.

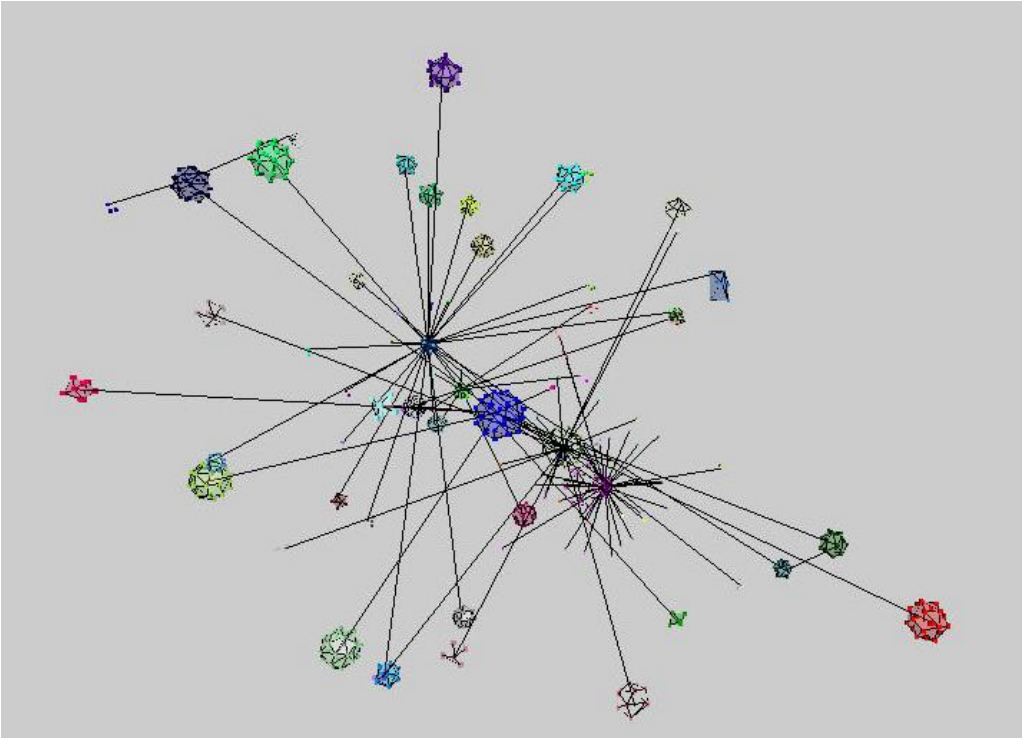


Figure 2: New placement algorithm results with hierarchy exaggerated

A small further modification to the algorithm yields even more legible results. The change is simply to weight each connection between vertices in the same group according to its similarity to the dominant node of the group. Thus the objects that most resemble the dominant node cluster 'close' to it, with those that barely made it into group membership lie at a distance.

Hull generation

Although we have experimented briefly with alternative solutions to highlighting regions of space (see figure 3) for one of the most exotic versions based on a 'vacuum packing' algorithm, the most practical solution has been to generate

polyhedron known as a *minimal convex hull* around the objects we wish to highlight. The problem of isolating points in a data-set that form a convex hull is well understood for two and three dimensions (and, indeed, can be mathematically extended to higher dimensions if needed). We briefly touch on some of the issues here[1].

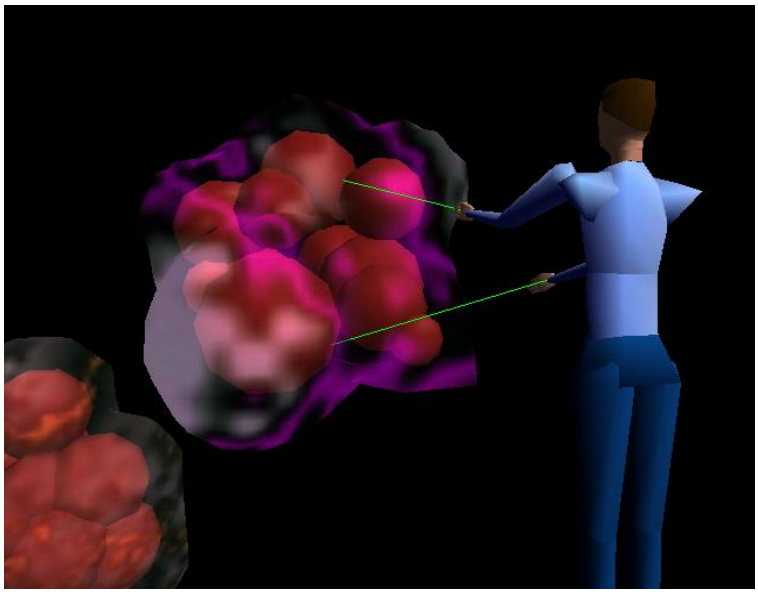


Figure 3: One of the more esoteric 'hulls' generated by 'shinkwrapping' objects in a collapsible mesh

1 for a more complete and thorough analysis of the problems see J. O'Rourke, Computational Geometry in C, Second Edition, The University of Cambridge Press, 1998, ISBN 0521649765.

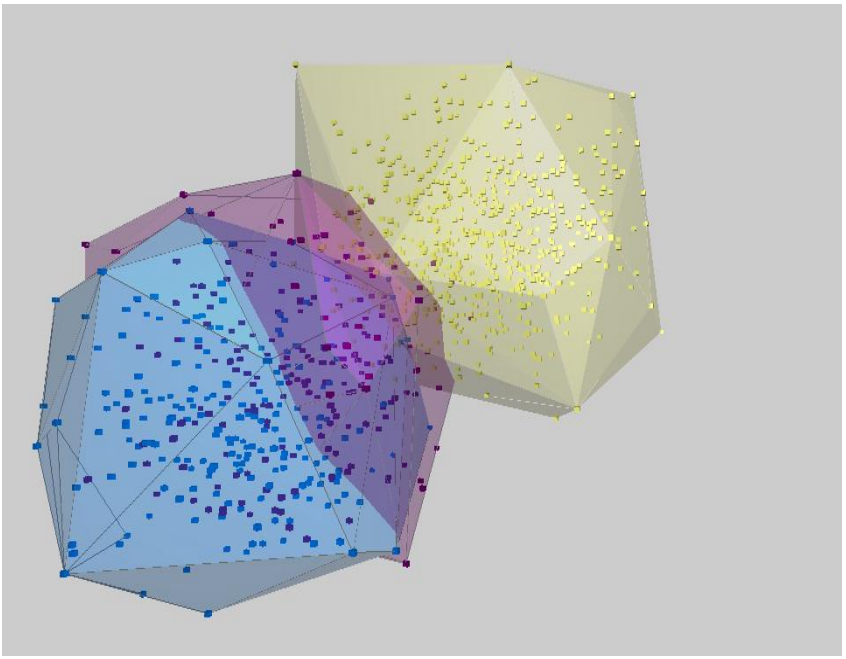


Figure 4: 2000 objects inside 3 dynamic hulls (framerate 30fps on 650Mhz PC with GeForce2)

An implementation in C is described by O'Rourke[1] This is the code upon which our hull generation routines have been based. Our previous implementation had the generation of the convex hulls taking part in the 'server' component, with the location of the vertices of the hull being downloaded into the 'viewer' component. In our re-implementation, the calculation of the hulls has moved into the viewer. This has the advantage that the hulls can be reshaped dynamically as the objects are moved in space. The lightweight nature of the hull algorithm that enables this repositioning was initially surprising. However on closer examination it became clear that two specific factors of the layout of the Q-PIT space made the real-time use of the algorithm possible.

1. The algorithm is order N^2 according to the number of objects that are to be included in the hull. Since previous algorithmic passes have already identified relatively small groupings of similar objects, it is on each individual group that the algorithm is executed (not the entire data space). Thus for a space with N objects divided into M groups, the cost of the algorithm is roughly of order (N^2/M)

[1] J. O'Rourke, Computational Geometry in C, Second Edition, The University of Cambridge Press, 1998, ISBN 0521649765

2. More significantly, although the algorithm is fundamentally quadratic, its performance varies greatly according to the form of the data. Consider two extreme cases, one where a large number of random points are positioned within a cube, and another where the same number of points are positioned so as to lie on the surface of a sphere. In the former example, the vast majority of the points do not lie on the surface of the hull and are eliminated rapidly from the search performed by the algorithm. In the latter case, the majority do lie on the hull, and the algorithms performance is severely impeded (a superlinear slow-down). For our hulls, which are generated around points laid out in space by the force placement algorithm, the majority of the points exist within the body of the hull, with relatively few forming the outlying vertices around which the hull itself is formed.

   There is good reason for this: for a point to be 'outlying', its similarity to the dominant node must be minimal. The previous stages of the Q-PIT algorithms work in such a way that the outlying nodes in a group are in the minority… anything less similar than these outlying nodes goes to form its own group. Thus the hull algorithm yields a high performance for the shapes we are generating.

Moving towards multi-user Q-PIT

Another aspiration of the eSCAPE work has always been to provide multi-user systems. Though the basic task of searching for a library book is initiated by a single user, and the Q-PIT technique itself is aimed at providing useful functionality for this work, we are examining the migration of the Q-PIT into a multi-user environment using the Deva VR system. The system itself is described in more detail in chapter 9 of this deliverable, as is the environment in which we are situating the multi-user Q-PIT. As well as providing an experimental platform for collaborative use of this browsing technique, the architecture of the Deva system provides a number of advantages for the basic Q-PIT functionality and it is these (rather than the issues concerning collaborative use of the browser) that we consider here.

A brief overview of Deva

The Deva system is aimed at large scale multi-user, multi-application virtual environments. In this capacity it provides services for processing the 'behaviour' of objects in the virtual environment as well as distributing the results of interaction to other geographically distant users over a network. The architecture of the system is described in detail in chapter 5a. Deva is designed to primarily use the MAVERIK kernel as its spatial management and rendering system, and

'properly' written MAVERIK objects can be included in a distributed Deva environment with relative ease.

Deva is an ostensibly client/server architecture, with a cluster of processors forming a 'parallel virtual machine' acting as a 'world server' and an arbitrary number of clients connecting across the network to visualise the contents of the worlds. A distinction is made between the 'semantics' or 'compute load' of an environment (which is managed by the server) and the 'appearance' (mostly rendering load) of the same environment (which is managed by the viewer clients). The Q-PIT application is a good example of this distinction and will be considered briefly here.

The Q-PIT has a number of tasks of various computational intensities.

1. Generating the q-space itself (communicating with the database, similarity calculation, minimal spanning tree, grouping decisions)
2. Force placement calculations
3. Generating the hulls based on the positions of the objects as they move with the forces
4. Rendering the results

The collection of algorithms in stage 1 above are heavyweight. Further they do not yield any meaningful results in real-time i.e. nothing sensible can be rendered until the chain of calculations has completed. This qualification is important, since it means that the algorithms, though vital to the generation of the space, do not themselves yield any perceivable or comprehensible results for the end user; it is the latter stages that make the former stages meaningful.

Stage 2 is an iterative algorithm, with increasingly good results that *are* presentable and meaningful to the end-user

Stage 3 turns out to be a relatively lightweight calculation that again yields meaningful results for the end-user.

Stage 4 is of course the most vital aspect for the end-user, since without the graphical rendering the whole process is meaningless.

Deva's architecture distinguishes between things that are perceivable to the user (these are located on the client) and things that constitute the semantics of the virtual environment (these are located in the server). It attempts to maintain a continuity of perception at all costs by moving any heavyweight calculation that might interrupt a smooth presentation into the server. Our usual example of this is that a user would perceive a cup falling from a table and spilling its contents, but would be unaware of the equations and calculation necessary to determine the effect of gravity and fluid dynamics on the environment. Though it makes the point, this example is somewhat artificial, since in reality the computational load of 'real' gravity and fluid calculations are beyond our current technology. Nevertheless, the distinction is an important one. The Q-PIT application provides a more concrete and compelling (though somewhat specialist!) example of this. In

our analysis above, Stage 1 constitutes the invisible semantics of the environment, and the remaining stages the perceivable components. Thus the Stage 1 calculations are moved to the server and performed within a thread of computation, whilst Stages 2, 3 and 4 are performed on the viewer-side to give 'smooth' visualisation.

There is no avoiding the fact that Stage 1 is a heavyweight computation that takes a number of seconds to complete. However this architectural split allows everything else in the virtual environment to continue being rendered smoothly and behaving sensibly as the heavyweight calculation takes place on another processor. The results of the calculation are then communicated to the viewer, which includes them in its force-placement, hull generation and rendering routines.

A second advantage of this architecture for the Q-PIT application is that since the server can be configured with a number of physical processors, each heavyweight computation thread can be allocated to a different machine; thus two Q-PIT queries issued at once from the same client may in fact be processed independently ('in parallel') on different processors.

The integration of a number of Q-PIT artefacts into 'The PLACE' is described in chapter 4 of this deliverable.

The Q-PIT in Action

This section describes the interface of the revised desktop version of the Q-PIT system. The 'immersed' version of the Q-PIT contained in the Placeworld is described in later chapters.

Use of the Q-PIT browser begins with the use being presented with a single window graphical user interface consisting of a number of controls arranged in the familiar 'menu bar', 'toolbar', 'workplace' and 'status bar' sported by most browser tools. Use of these controls will be illustrated in this chapter through a fabricated user session.

The browser is controlled by two three input devices; a traditional keyboard for entering keywords, a regular two buttoned mouse for selecting objects in the 3D space and for controlling the 2D widget set, and a regular two buttoned trackerball (in our case the Logitek Marblemouse) for navigating in the 3D space.

A session begins by the user typing in a keyword in the field located near to the top of the browser pane. At this point the browser connects to the database engine and initiates a query (which could be fielded by the engine to Lancaster University OPAC, amazon.co.uk etc). The status bar at the bottom of the pane indicates the various stages of handshake that occur between browser and database engine, to give the user an indication that 'something is happening' whilst the download of the results takes place. At the same time, a 'progress meter' indicates the percentage of queries that have been downloaded. The response time of the server varies depending on whether or not the keyword has been used recently (in which

case results come from its local cache and take a small number of seconds to be returned; alternatively the results are retrieved from the website or OPAC database, resulting in a longer wait of the order of tens of seconds).

Once the results have been retrieved, a small cluster of objects all positioned on top of one another appears in the 3D space directly in front of the user and at a fixed distance from their eye-point. A further short wait (with status and progress indicators) is encountered as the similarity calculations are performed. At the end of the period of calculation, the q-space 'explodes' into a formation in real-time, with the linkage and hulls being calculated on each frame to give the user a first impression of the layout of the space.

We first consider a keyword search that has returned a small enough number of objects for the recalculation of weighting to be performed in near-realtime (for the hardware of the current system – a 750Mhz PC with a GeForce graphics accelerator) this means less than 200 objects. For such a number of objects the force placement algorithms converge to a quasi-stable solution rapidly, and an essentially 'static' molecule formation is displayed, with little perceivable motion in the objects. However, since the algorithms are still active, any interaction with the objects via the mouse (described later) or by modifying the controls of the interface causes a real-time relocation.

The browser supports two modes of navigation in 3D space using the trackerball. The first is 'free' navigation, the current mode being indicated by the central icon in the 3D space. For free navigation, the icon is four outward facing triangles arranged in a square, for 'object centric' navigation, it is a circle. The browser begins in free navigation mode. When in this mode the rotation of the trackball corresponds directly to the rotation of the user's viewpoint; the two axes of the ball corresponding to 'pitch' and 'yaw'. The two buttons on the trackball cause forward and backward motion of the viewpoint along the view vector. Essentially this allows the user to roam the 3D space, unconstrained (and also unaided!) by its semantic contents.

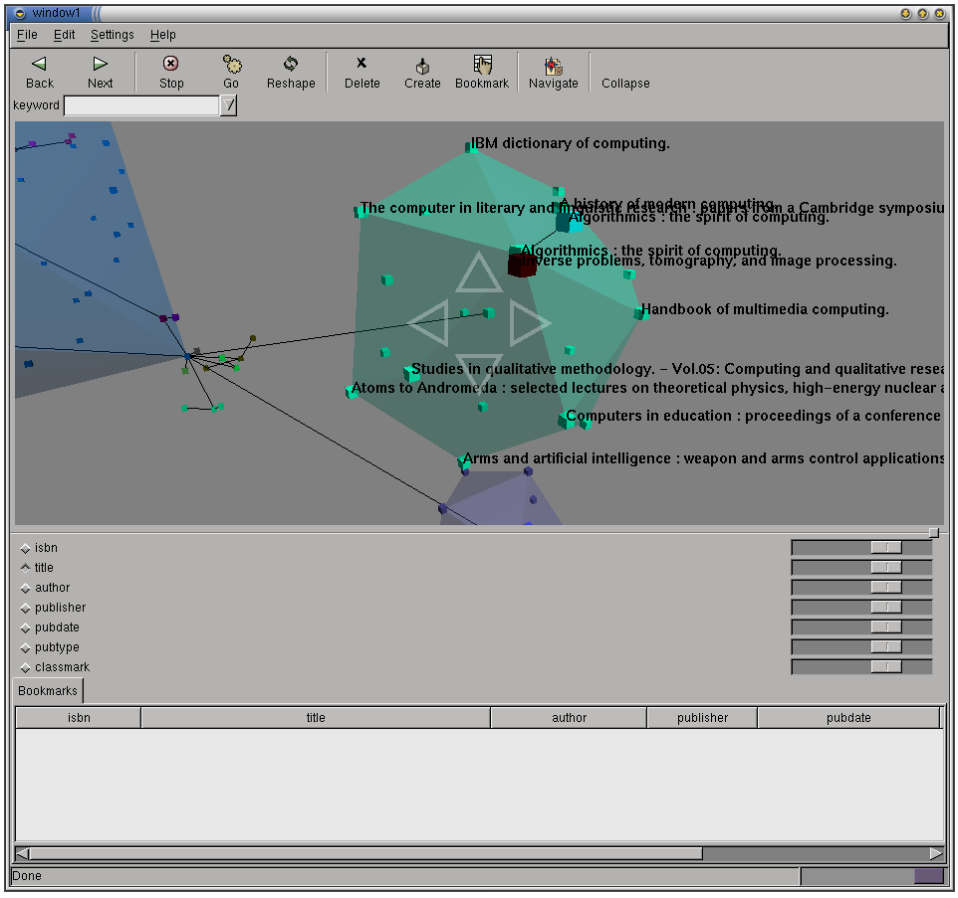


Figure 6: The q-pit browser window in 'free navigation' mode (triangular navigation icon)

At this stage the user is presented by a multi-coloured molecule like structure consisting of objects (representing the tuples of the database), hulls (representing groups of similar objects) and links (representing relationships between the objects and hulls). Nearby objects are labelled with descriptive text. This text is determined by the currently selected field in the centre pane; by default this is the book title, however it may be altered to display, for example the author or ISBN of a book, using the radio-buttons against the relevant field in the centre pane.

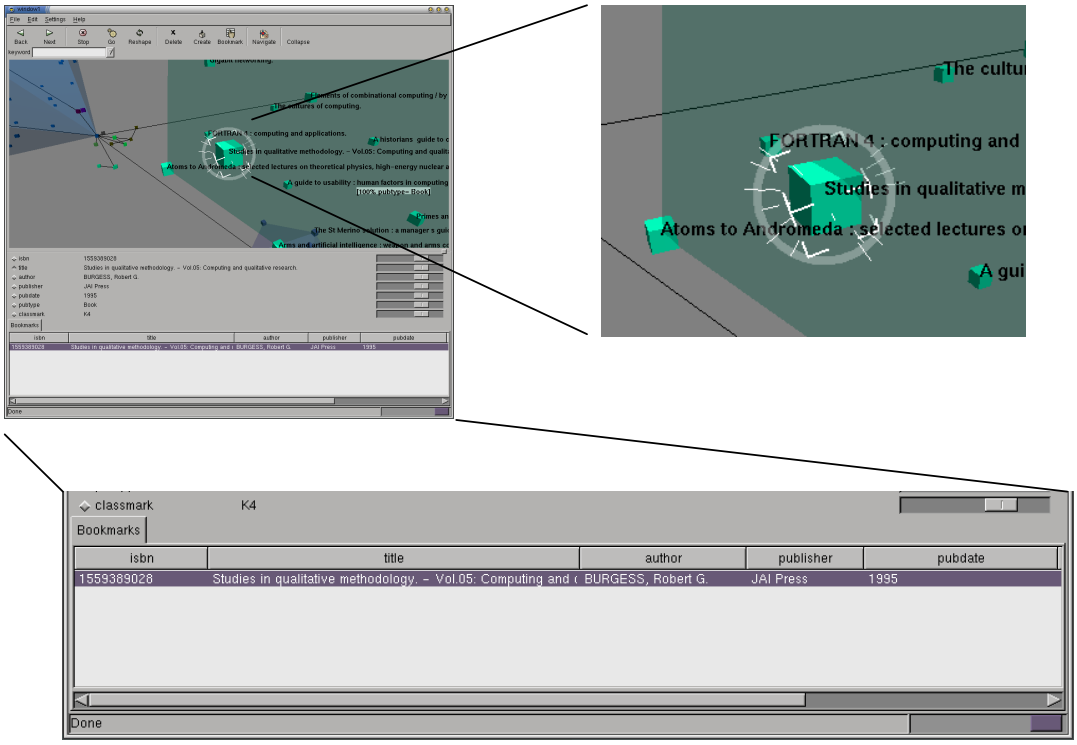
With the mouse the user is able to inspect the various artefacts in the 3D space; leaving the mouse momentarily over an artefact causes a 'tooltip' like description to be displayed. For a book object, the description is the 'major' field of that book (e.g. a title); for a hull it is an attempt to summarise its contents (e.g. '100% published in 1999, 50% pamphlets').

Selecting an object

Objects are selected primarily by pointing at them with the mouse and clicking with the left button. This has four effects:

1. The user's viewpoint is moved smoothly from its current position to that of 'orbiting' the selected object at a fixed initial distance. The object becomes the centre of attention and of view.
2. The object is 'selected', and a rotating 3D cursor box is positioned around it to indicate this.
3. The user's navigation mode is changed from being 'free navigation' to being 'object centred' navigation. Until the mode is further modified, movement of the trackball now causes the user's viewpoint to orbit the selected object along two axes of rotation. The user is able to zoom up towards or away from the object, but it will always remain in the centre of the scene. To indicate this change of navigation mode, the 'triangle' icon is replaced with a circular one as shown in figure 7. Returning to free navigation is achieved by selecting the 'navigate' button from the toolbar.
4. The full details of the object are displayed in the centre pane of the browser.

Two operations can be performed upon a selected object. These are initiated by clicking on the buttons contained in the toolbar at the top of the browser window.



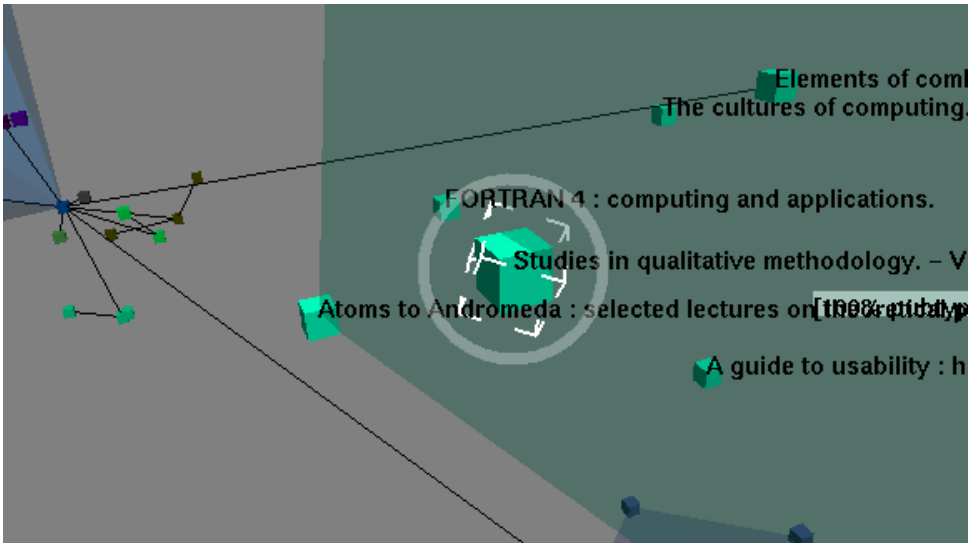

Elements of
The cultures of computing.
FORTRAN 4 : computing and applications.
Studies in qualitative methodology. – V
Atoms to Andromeda : selected lectures on
A guide to usability : h

Figure 7: Navigation changes to the 'object centric' mode which is indicated by the circular navigation icon

Bookmark

This function copies the information from the selected object into the scrolling list of 'favourite' items at the base of the window. Objects in the bookmark list can be revisited at a later date by double-clicking on their entry in the scrolling list. Bookmarking an item causes it to be highlighted by a 'sparkle'.

Delete

This function removes the selected object from the 3D space. If the space contains a small enough number of objects, the deletion of an object causes a recalculation of the similarity matrix and subsequent tree and groups as described in the previous chapter. Consequently, the 'best position' for the object as determined by the force placement algorithm and the 'group' configuration represented by the hulls may change, and this is indicated by the objects leaving/joining coloured hulls and moving in real-time to their new best positions. For larger configurations of molecules this dynamic reconfiguration upon delete can take a significant amount of time, and can be disabled through an option in the menu bar (a warning dialogue appears if the object being deleted is a member of a large query, to inform the user that they might wish to disable the automatic reconfiguration option rather than face a wait each time an object is deleted).

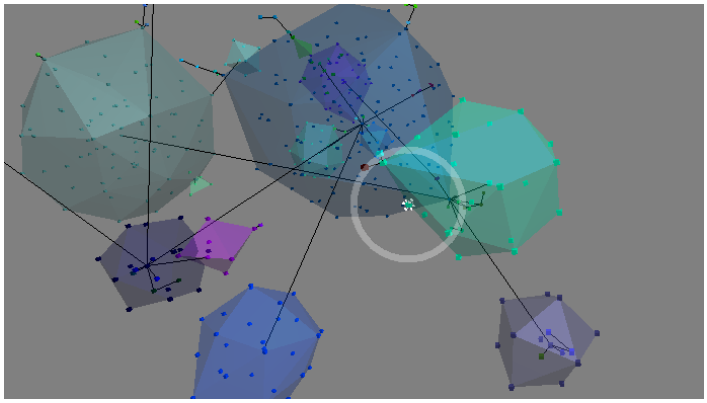


Figure 8: Initial configuration of the q-space

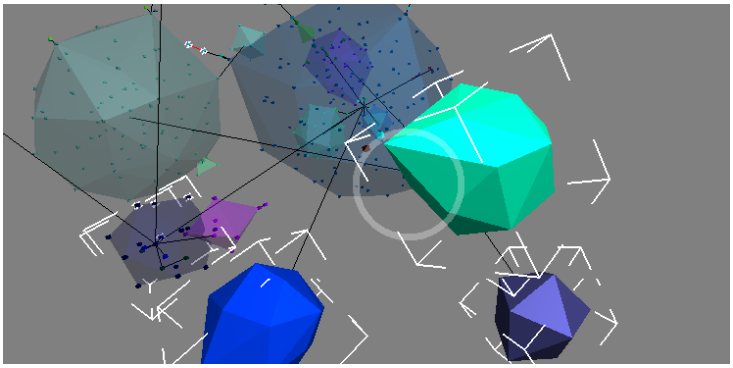


Figure 9: A number of artefacts (hulls and objects) are selected for deletion

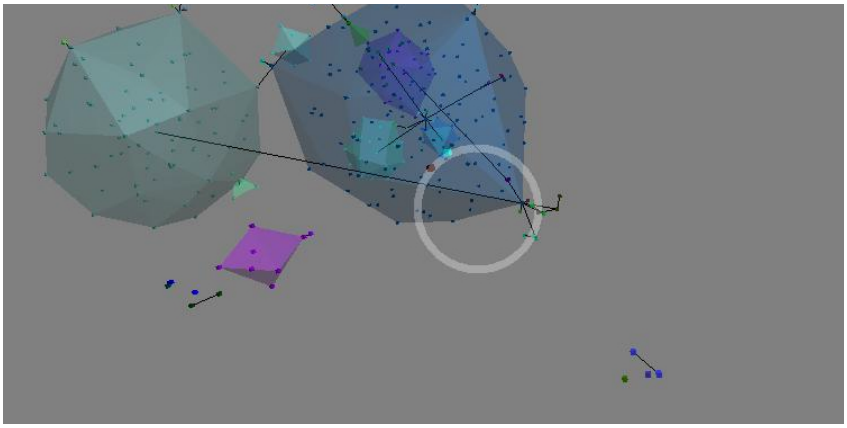


Figure 10: On deletion the artefacts vanish from the space

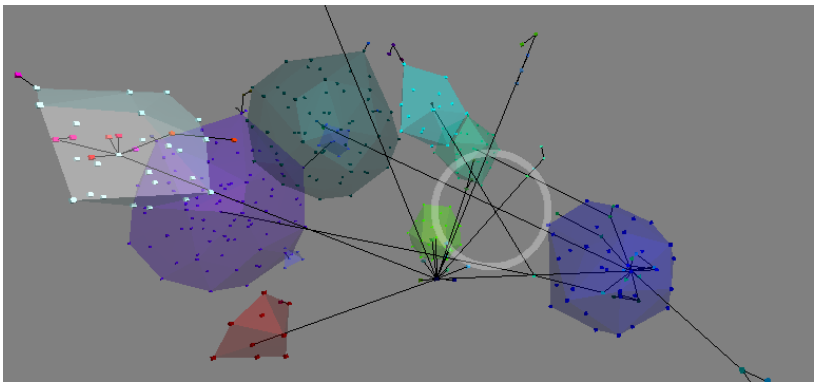


Figure 11: Finally the space reconfigures itself

More forms of selection

Selecting hulls

As well as selecting objects as described above, whole hulls may be selected. Apart from highlighting the hull as being a selected artefact, this toggles the transparency of the hull, allowing its contents to be hidden if the space is

becoming cluttered (additionally the hiding the contents of hulls provides a convenient means of reducing the rendering load of a molecule and allowing larger molecules to be browsed at improved framerates).

A hull is selected only if it is the only artefact in the line drawn between the user's viewpoint and the mouse pointer. If an object exists on this line, even when occluded by multiple hulls, it will be selected in favour of the hull.

A selected hull implies selection of its contents; thus if delete or bookmark operations are performed whilst a hull is selected, the operation is applied to all its contents.

Multiple selection

Selection of multiple artefacts is possible by holding down 'shift' on the keyboard whilst selecting them with the left mouse button. Hulls may be selected as well as objects using this technique. Selection whilst holding down shift does not move the user's view point, neither does it change from free navigation to object-centric navigation mode or display the object's details in the centre pane. This function allows large number of objects and hulls to be selected simultaneously for deletion of for bookmarking.

Reconfiguration of the molecule using interest weights

Along side the various fields of the objects in the centre pane are a series of slider controls that indicate the user's current preferred weights for the fields. Altering these weights using the mouse causes a dynamic reconfiguration of the hulls and positions. Again, this dynamic feature becomes problematic when a large number of objects are present and the similarity calculations become heavyweight. An option in the menu bar allows this feature to be disabled.

Manual control of the reconfiguration

Three buttons on the toolbar allow the user to intervene in the algorithms that reconfigure the q-space. For small molecules, little intervention is necessary since the algorithms are processed in real-time and the changes to the molecules occur rapidly. For larger spaces however, both the calculation of the similarity matrix and the processing of the force placement algorithms become more time consuming.

Halting and restarting the force placement algorithm

Once a reconfiguration has been initiated, the 'best' position for each object in the space changes, and the force placement algorithms iteratively calculate new positions for all the objects in order to move them towards this solution. However, for a complex q-space there is no guarantee that a 'stable' position can be reached. Also, the point at which a 'good' solution is reached is a very subjective experience and will depend on the particular likes of the user and the task at hand

as to when such a solution has been obtained. The ‘stop’ button on the toolbar allows the user to indicate either that they are not willing to wait for the solution to be reached or that a good enough solution has already been reached for them to begin navigation of the space. Activating this control interrupts the force placement algorithm at its latest iteration, often providing a significant improvement in the interactive performance of the browser.

The ‘go’ button on the toolbar resumes the calculation from its previous position.

Forcing reconfiguration of the space

For large queries where the performance of the browser is impeded by the repeated dynamic reconfiguration of the space on deletion or alteration of the weights, the ‘reshape’ button on the toolbar forces the algorithms all to be re-applied. When the browser is in manual mode, the algorithms are dormant during deletion and change of weights, and are reactivated when the reshape button is pressed. The reshape button also causes the force placement algorithm to reactivate if it has been halted using the ‘stop’ function. Reconfiguration of space is implicit when issuing a new keyword search.

Finding similar books

Double clicking on a selected book object causes a new ‘inner’ space to be created that contains all books from the current database that share the same classmark. This inner space is treated to all intents and purposes like the larger outside space surrounding it, with the exception that for the current demonstrator it is not meaningful to recursively select books (since an identical set of the same classmark would be returned).

An ‘exit’ button appears on the toolbar to allow the user to leave this space and return to their previous position in the main query space.

History Trails

As the user moves around the q-space selecting objects, a trail is left behind that passes through all uniquely selected books. The trail follows the books as they move from hull to hull, and detach themselves automatically from deleted objects; thus trails represent the semantics of the history rather than the spatial relationships of navigation. The ‘back’ and ‘next’ buttons on the toolbar move the user smoothly from their current viewpoint to the appropriate position on their history trail, bringing back into focus the object at that position in their history.

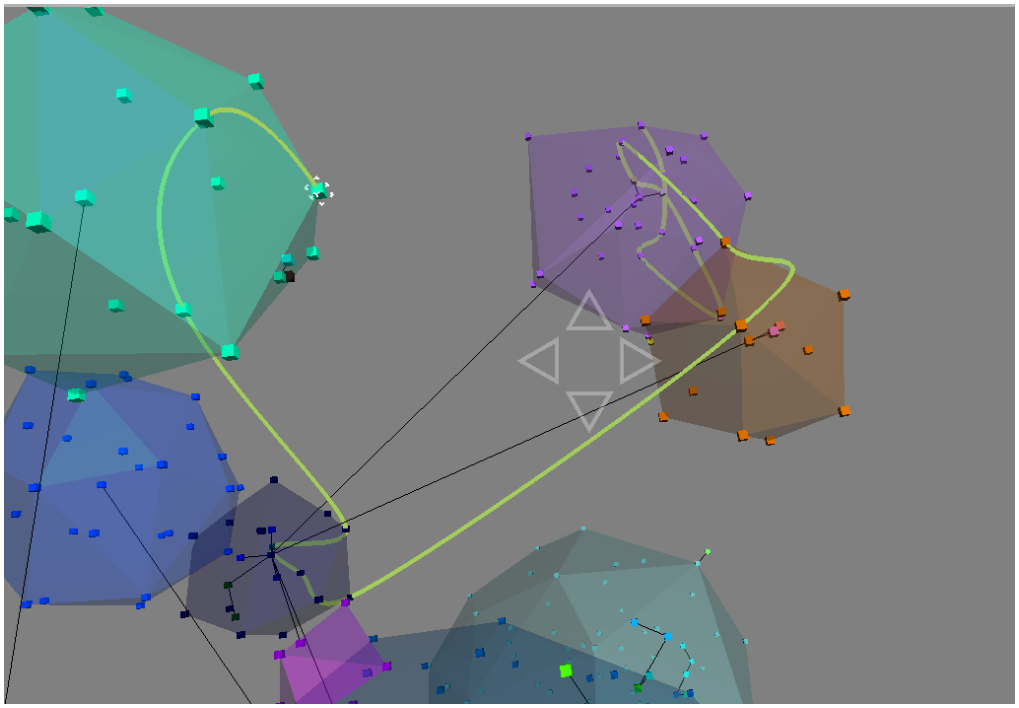


Figure 12: History trails

Addressing the issues from the first set of studies

In this section, we revisit the issues from the first set of studies and briefly indicate how we have addressed them in the new implementation of the Q-PIT system.

Constructing a reliable demonstrator that does not 'crash' when confronted with real world tasks.
The reimplementation of the system using MAVERIK and Deva has side-stepped this problem.

Scaling the demonstrator up to handle large searches (+350 objects).
Out current implementation running on a 750Mhz PC with GeForce graphics accelerator can manipulate and render between 1000 and 2000 objects at interactive rates.

Furnishing users with the full title of objects so that they may conduct real world searches for information.
A technical limitation of the initial prototype that is not present in our new implementation

Reducing inconsistencies in colour usage.
The 'hull synopsis' to some extent alleviates this problem, by giving users an indication of what a hull contains, thus making it clearer that there is not a direct between the properties of the object in the database and its representation in the virtual environment. That is 'not all blue objects are books' for example.

Supporting topic-based searching by enabling users to browse objects that share the same classmark from any point in the course of searching results.
Double clicking on an object now generates a new space containing books that have the same classmark as that selected.

Supporting navigation in 3D space in providing a 'centred' view.
Our new implementation generates the query space at a fixed distance into the user's current viewpoint. The initial formation of the molecule is therefore always at least partially visible to the user.
Providing more flexible means of navigation to allow the more effective manipulation of specific objects located within data clouds.

Supporting navigation in providing an efficient aesthetic layout of the visual field.
The redesign of the interface reported in chapter 4 shows a new interface that seeks to improve the aesthetics of the visual field.

Enabling searching to proceed by a process of elimination.
A 'delete selected objects' allows rapid removal of individual objects or groups of objects to enable searching by elimination

Providing more legible or readable object text displays.
Rendering the labels as 2D text projected from the 3D space provides considerably more legible text

Providing a history function so that users may retrace the steps taken during any particular search
Provided via the history trails and the history navigation controls

In the following chapter we consider the re-calibration of this amended version of the abstract electronic landscape through a series of studies undertaken with

users. This reassessment of the demonstrator explores the utility and success of the amendments made to the developed environment.

Chapter 8: Calibrating the Library Demonstrator

Andy Crabtree, John Mariani, Steve Pettifer, Andreas Kratky
Lancaster University, The University of Manchester, ZKM

This chapter reports the ongoing situated evaluation of the library demonstrator following the changes made in response to last year's user trials reported in the previous chapters.[1] As described in Deliverable 4.0 and 4.1, the strategy of evaluation is directed towards elaborating technical designs through confronting computer-based artefacts with end-users and practical circumstances of use. In line with more traditional forms of 'usability' study, situated evaluation is an observational exercise oriented towards the performance of usability trials. Specifically, to the ways in which users accomplish the activities that they are asked to engage in; to the practical problems they encounter in doing those activities; to the confusions that arise in the doing; and the solutions devised to make the technology work *in situ*. As an integral feature of the common methodology, situated evaluation is a means of 'calibrating' technological configurations to meet the practical needs of end-users (A more complete overview of this is provided in D4.3. Common Methodology). Below we report this year's evaluation, the problems identified by users in confronting the technology, and the technical solutions which may be implemented to meet the needs of a real world community of users.

The library demonstrator in use

The following (edited) sequence of talk was preceded by a brief introduction to the library demonstrator by the ethnographer (Andy) informing the users (Chris, Suzy, Penny and Tom) as to the purpose of the trial, namely to assess the practical adequacy of the library demonstrator in accomplishing topic-based searches. The users were informed that Steve (one of the designers) would first do a 'walkthrough' to show the users what the demonstrator is, what it does, and how it works, and would then invite each user to 'have a go' with his support. The 'walkthrough', in which *all* functionality was demonstrated, will not be described here, as the organisational features of such activity, and its relevance to design, has been dealt with elsewhere.[2] Here we concentrate on user interaction with the

---

[1] Crabtree, A. (1999) *Situated Evaluation of the Demonstrator*, The Library Abstract eSCAPE Demonstrator (eds. Mariani, J. & Rodden, T.), eSCAPE Deliverable 4.1, p. 109-146, Esprit Long Term Research Project 25377, ISBN 1 86220 079 3.

[2] Crabtree, A., Pettifer, S., West, A. (1999) *Inventing new technologies: the economics of information and situated evaluation*, Towards a Common Methodology (eds. Hughes, J.A., Crabtree, A., Rodden,

new library demonstrator. Users consisted of administrative staff, tutors, researchers, and students. We have chosen not to report separate user 'drives' as we did last year but are instead selective, using one sequence of talk that elaborates the socially organised work common to each encounter.[1] We have chosen this approach to reportage as user reactions were much more uniform in this second trial, largely as a result of the changes made in light of the original trial – simply put, fewer 'troubles' were experienced. Following reportage of the work of the trial we review emergent interactional topics for consideration in design, report 'good ideas' for future design from users, and assess the overall adequacy of the demonstrator as seen and understood from a practical point of view.

#1. Getting a real feel for the space

Chris: I've got three ideas that I'd like to do. It depends on the time as to
Steve: Okay. So if you sit there (Steve moves seat, gives the user access to the demonstrator).
Chris: (Chris assumes position at keyboard, puts one hand on the ordinary mouse) Okay. Keyword.
Steve: Actually, if you put your hand on that thing there [the 3D mouse control] and just get a feel for it. So that moves you in and out (Chris is clicking on mouse buttons).
Chris: Yes.
Steve: And roll the red ball around.
Chris: Umm. Right. So that's just the basics.

#2. Issuing a query

Steve: Right, just find yourself a blank bit of space if you want to make a new query.
Chris: (Moves to a blank area and types keyword into query box).
Steve: Okay, and just press return.
Suzy: What've you typed in?
Chris: Cookery
Steve: (Demonstrator returns 'hits'). Oh, that's disappointing
Chris: Not many books
Steve: Thirteen. It say's down the bottom there 'returned thirteen'.
Chris: That's interesting, okay, right.
Penny: We don't really have a cookery department or anything do we.

#3. Navigating the q-space

Chris: (Starts to navigate hits). So I'm zooming in now
Steve: That's the red ball, yeah, to point yourself in the right direction
Chris: And I'm wanting to go right in.
Steve: So that's probably the left button on that
Chris: Oh yeah – ahh.
Steve: If you can't do it with your left hand you might find it easier to move it over onto the right and use it with your right hand.
Suzy: It's disappeared. (Chris has navigated right into the centre of a data cloud).
Chris: It's disappeared!
Steve: It's behind you. There it is. Press the (points to zoom out button).
Chris: Oh I see. I'm with it.

T.), eSCAPE Deliverable 4.0, p. 85-106, Esprit Long Term Research Project 25377, ISBN 1 86220 078 5.

[1] For a deeper appreciation of the methodological election here see Garfinkel, H., Lynch, M., Livingstone, E. (1981) The Work of a Discovering Science Construed with Materials from the Optically Discovered Pulsar, *Philosophy of the Social Sciences*, vol. 11, p. 131-158.

#4. Working with the q-space

Steve: It all makes sense now. Okay, so what's happened at the moment is that you've got – down here, this little list of diamonds here [the weightings list]
Chris: Yeah
Steve: Shows you which of the – press stop for a moment
Chris: Stop.
Steve: That'll stop this thing [the data cloud] from sliding around in space
Chris: (Presses 'stop' button on toolbar).
Steve: That's it.
Chris: Right, now I want

#5. Using weights

Steve: Now, at the moment you're showing the ISBNs for those which is probably not what you want
Chris: Yes and I don't want that, I want
Steve: You want to select
Chris: Author. (Looks again at weightings list) I think I'll have title.
Steve: Right, so click there.
Chris: Yes I understand, right, okay (clicks on weighting button).
Steve: That's it.
Chris: And I want that to be important
Steve: Right, and everything else probably not
Chris: (Puts other weights at zero).
Steve: Right, and press 'reshape'.

#6. Browsing the data in a data cloud

Chris: Reshape. Okay, so I want to now – oh (the data reconfigures itself according to the re-weighting; Chris starts to browse the reordered space). So, now, the cottage economy and cookery, wartime cookery book, kitchen front, fine, okay, how to use yeast, guide to wartime
Penny: No Delia Smith in there
Chris: Ah, no Delia, no Delia
(Everyone laughs)
Suzy: No Naked Chef either is there

#7. Preparing to issue a new query

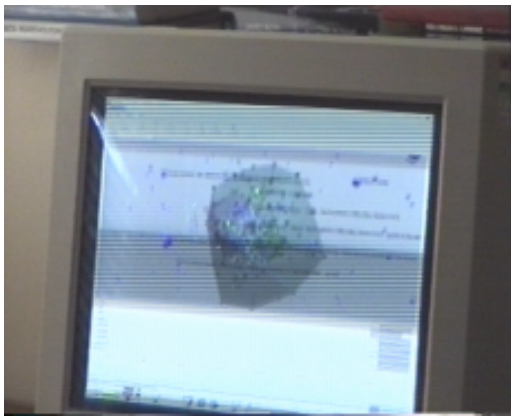
Chris: Drying and Salting, that's an interesting selection of books isn't it. Okay. Now I want to just abandon that, so I can just wheel that out of the way
Steve: Yeah
Chris: And then try another, is that right?
Steve: Are looking for a completely different subject?
Chris: Yes.
Steve: Yeah, fine, in that case just wheel it around.
Chris: (Moves view to a blank bit of space and types in new keyword).

#8. Issuing a new query

Steve: Research administration, press return.
Chris: That should be interesting.
Steve: I've no idea
Chris: Although, well, we'll see
(Waiting for hits to be returned from OPAC – approx 15 seconds)
Suzy: It's having a slow think this time isn't it.
Steve: Umm. Well that could mean that there's lots of books to come back. W'hup, four hundred. Quite a lot of them.

#9. Preparing to navigate a multiplicity of data clouds

Chris: Right, okay, and how many, this, oh my goodness gracious (Chris is reacting to the sudden population of the space with four hundred objects).
Chris: Right, so we just wait for it to just
Steve: You can just wait for it to settle for a bit. If you zoomed out you'd get a better view of what it was doing whilst it's settling
Chris: Oh, right. Okay, zooming out (Chris tries to zoom out)
Steve: It's probably the right-hand side [the zoom out button on the mouse].
Chris: Oh, yes, it's getting just used this. Fine. Oh right.
Andy: That's a lot of books.
Chris: (Starts browsing the data clouds). It might have taken research as the
Steve: Press reshape just to be on the safe side. Make sure that that's according to the criteria that you want. Hopefully it won't reshape it [it doesn't].



#10. Establishing the meaning of data in a multiplicity of data clouds

Chris: Right now I want to actually zoom in on this pink [cloud] there.
Steve: If you put your mouse cursor over it, it might give you some indication of what that pink cloud is all about. (Chris does that).
Chris: Over-size pamphlet, okay. (Chris clicks and zooms in rapidly on a particular object in the cloud).
Steve: So what you did there, because you clicked on the object, it zoomed you in on that specific book.
Chris: Right.
Steve: You might want to back off a little bit.
Chris: Yeah, okay.
Steve: I'd also press stop at this point as 'cause things are still gently drifting about.
Chris: Yeah, it's just navigating round – umm. (Chris presses stop and resumes browsing the clouds).

Andy: So you want to be able to find out what those clouds are as it were.
Chris: Yes, yes. So say I wanted to go this one
Steve: Now currently you're still locked in on that book that you
Chris: Yes, so I'm wanting to
Steve: You want to escape so press the 'navigate' button [on the toolbar].
Chris: Yeah, okay, right. I'm still in stop?
Steve: Yes. Stop just means that the space isn't shifting around, all the objects will stay still.

Steve: So, if you want to actually fly towards one of those things [the data clouds] you'd have to select one of the books inside it which might be difficult from this distance.
Chris: Right.
Steve: You can probably just about do it if you
Chris: (Clicks onto a cloud).
Steve: Click again 'cause you selected the hull there.
Chris: Yeah.
Steve: Now move the cursor round until it gives the details of a book.
Chris: (Clicking on data cloud).
Steve: Just try and point to one of those little dots.
Chris: Oh right, I see, yeah.
Steve: And as soon as it gives the details of a book rather than the details of the hulls
Chris: (Trying to move cursor over one of the objects in the cloud).
Steve: Yeah. Now if you'd clicked when it said John Mariani there
Chris: Oh, okay.
Steve: Then you would have flown towards that object. Alternatively, you could just use the fly forward button which will get you a bit closer.
Chris: Yeah (Trying to point to an object again). Oh goodness!

Chris: What's this thing [data cloud] doing behind here. How can I get rid of that?
Steve: Just click on it again.
Chris: (Clicks on the cloud again).
Steve: What you're doing there is, you're selecting that whole group.
Chris: Yes, and I want to get away from that
Steve: You want to select an individual book.
Chris: So
Steve: So wait until it gives you the title of the book. There you are, that was the title of the book.
Chris: Oh.
Steve: Point, point it at one of the little dots.
Chris: (Laughs). Yeah, okay.
Steve: So that's the hull you've selected there rather than the
Chris: Yes.
Steve: So move the mouse – point
Chris: Oh (moving mouse).
Steve: That's it. Now click. That's it (zooms in on object)
(Everyone laughs)
Chris: Oh thank goodness for that.
Andy: You've actually got the principle there. You just click on object within a cloud and you can see what the cloud's about and then you can dump that cloud if you don't want it.

#11. Browsing a multiplicity of data clouds

Chris: Okay. So I now want to go, zoom to this one.
Steve: So still locked in around that object.
Chris: Navigate, okay. Unrelease it. Uh – I don't want that one (Chris has selected the hull of a cloud).
Steve: You've got to click on an individual book to go towards it. So click on the cloud to deselect the cloud first. Just click on that great big blue mass.
Chris: Yeah.
Steve: That's fine, and now if you click on individual objects you will fly towards them.
Chris: (Clicks on an individual object).
Steve: That's right.
Chris: (Chris is browsing individual objects in a cloud) Health research programme, foundation of [inaudible] research, okay.

Andy: So what kind of thing have you got there, all health or
Steve: Yeah, they've got medical kinds of things by the looks of it.
Chris: Okay, so I want to fly out of this. I've clicked on that so

#12. Deleting data clouds from the q-space

Steve: Is it that you're not interested at all in those?
Chris: I'm not interested at all.
Steve: Okay, so zoom out until you can see the blue hull.
Chris: (Zooms out).
Steve: Right, now click on that hull, and press delete.
Chris: Yes, I see, delete (clicks on the delete button on the tool bar).
Steve: And now press reshape.
Chris: (Clicks on reshape button on toolbar) And reshape.
Steve: So now it's reshaping the space as if those books never existed.
Chris: Okay.
Steve: You should probably zoom back out a little bit.
Chris: (Zooms out) Right, okay.

#13. Refining the search in browsing and eliminating data clouds

Chris: So I need to now – I want to find – it's still floating so if I want to just stop (clicks on stop)
Steve: You're still locked in an object.
Chris: I am – navigate (clicks on navigate) yes, okay.
Steve: If the cursor in the middle is a circle that means you're still focused.
Chris: Yes, okay.
Steve: And if it's those four pointy arrows then you're free to go anywhere you want.
Chris: Okay. (Browsing the data clouds; trying to point to individual objects so that titles can be read).
Penny: It's sort of frustrating because it's slow at zooming.

Chris: Right, I'm trying to head for this brown blob here.
Steve: You don't have fly by hand of course, you can just select an object
Chris: Just select this (gestures to objects in cloud)?
Steve: Yeah, if you just point at one of the
Chris: Like that? (Chris has selected the hull of the cloud).
Steve: No, one of the objects in that cloud. Just select one of the books.
Chris: Oh, I see. (Deselects cloud, selects an object in it, and reads title). So I'm not interested in that at all, so - and I'll delete.
Steve: Yeah.
Chris: Delete.
Steve: And now press reshape.
Chris: And then reshape, okay, right.

Chris: (Browsing remaining clouds). Don't want that either. (Deletes and reshapes).
Chris: (Browsing) Ah, here we go, we're getting there now I think. No, research methodology. Research methods
Steve: Doing research with children
Chris: What I'm after is framework five. That's what I'm wondering – how to find that within
Andy: So basically you've got to a point where - 'cause you've refined your own query a bit more now, you might actually articulate that.
Chris: Uh uh
Penny: What, put in a new keyword?
Andy: Yeah, just like you would do in the library.
Penny: Right.
Chris: Right.

Emergent features and issues for design

The talk above makes available sequential orders of action which vary according to the kind of Q-space produced in response to a query. The user trial was organised by participants through one of two basic sequential orders of action. One undertaken in response to a small number of returned objects (#2 to #7), and the other undertaken in response to a large number of returned objects contained in a multiplicity of data clouds (#8 to #13). Below we consider the organisational features of the respective sequences of action and (thus) identify interactional topics emergent from user interaction with the new library demonstrator and their relation to design.

Sequence 1.

Getting a feel for the real thing. All user interaction in the trail began with a course of basic navigation where each user got their 'hands on' the navigation device in order to 'get a feel for it'. The point and purpose of this work was to learn the basic controls: two buttons on the mouse zoomed the user in and out and a track ball let them rotate and move up and down. This new navigation device was implemented in the attempt to provide more flexible means of navigation and was well received by users, being simple and easy to learn. Further improvements could be made from a user perspective however. Specifically, while the new device supported ease of use in navigation another device (an ordinary mouse) had to be used to work on objects in the Q-space. Users found this confusing and occasionally frustrating. Ultimately *one* navigation device incorporating the functionality of both existing devices would be preferable from a users' point of view.

Issuing a query. Following familiarisation with the navigational controls, users issued queries. The mode of issuing a query was intuitive and required no instruction. Users did have to be instructed to move to a 'blank bit of space' when issuing a query. This is not an issue of current concern but would need to be addressed along with issues here (such as awareness of hits returned) in developing online engagement sequences supporting use of such electronic landscapes by novice users.[1]

Navigating the Q-space. Work here was directed towards developing a working familiarity with the navigational controls. That is, learning how to use the controls in practice. All the users felt that while 'tricky' to use at first they would soon 'get the hang of it'. The practical issue was simply one of coming to terms with a novel piece of equipment that is, in itself, essentially simple to understand and use.

[1] For a discussion of relevant issues here see Crabtree, Pettifer, & West (1999).

Working with the Q-space. In order to undertake work in the Q-space users had to learn new functionality. Specifically, the 'stop' function which stabilised the molecular elements of the emerging Q-space and made them amenable to being worked on. The functionality is easy to learn and readily accessible in being visibly available on the toolbar. Any issues here go towards the development of an engagement sequence for novice users.

Using weights. The concept of weights was seen as troublesome in the original user trial but experiences in the second trial indicate that the notion of weights is less problematic than originally thought. Perhaps this has much to do with the new layout and presentation of weight functions. Whatever the case may be, the weight functions were well received and represent a clear success. Nonetheless improvements could be made. At an ergonomic level, the weight sliders could be placed closer to the weight categories, reducing hand-eye coordination in manipulating weights. More significantly, users often found the returned Q-space confusing and rather meaningless. Although a degree of basic 'structure' was provided, the assumption was that the user will 'reshape' the Q-space according to their preferences. Users have assumptions of their own, however, and expect the Q-space to be a meaningful space in the first instance. Naturally we want to retain flexibility yet need some workable solution. One potential solution is to have the default weighting set up as 'classmark-as-topic' like in the original demonstrator. That is to say that objects will again be grouped by default according to their classmark membership. The classmark will not be displayed however, rather titles will be. This allows users to get a sense at-a-glance what any cloud is 'about' thus providing users with readily intelligible topic spaces. Such support has been reimplemented in the context of library searches.

The concept of grouping lies at the core of the Q-space and this allows us to consider the development of a generic approach to grouping. The general algorithm exploits attributes associated with information entities. These entities can be added to a wide range of information. This allows the general Q-space engine reported in chapter 10 to group information drawn from a wide range of sources. The grouping also allows us to present the information using both 2D and 3D interfaces. Users can select the interface and explore the grouping in the manner most suited to the information device used to access the general Q-space. In the following sections we focus on the facilities provided by the full 3D Q-space interface.

Browsing data in a data cloud. Following the reshaping of the Q-space, users browsed the contents of the data cloud, reading the full titles of objects in making a decision as to the data's relevance to their search activity. Although demonstrated during the walkthrough, no users employed the history function on the occasion of the trial. All users thought the history function useful and suggested that it may be extended in the fashion of the Internet, enabling them to examine histories over several weeks.

Preparing to issue a new query. As with learning to use the navigation control, work here was directed towards learning the actions required to issue a new

search, which simply consisted in moving to a 'blank bit of space'. Simple enough, although in practice it does beg the question of 'clearing' the space and making that functionality plainly visible.

Sequence 2.

As before, the user issues a query unproblematically but this time is presented with a large number of 'hits' which are arranged into a multiplicity of data clouds.

Preparing to navigate a multiplicity of data clouds. Some instruction is required from the designer and the user zooms out accordingly in order to see the new Q-space. No significant difficulties were experienced here although it might be desirable to implement an automatic zoom that enables the user to view the entire space as the size of the Q-space dictates, rather than having to take manual action.

Establishing the meaning of data in a multiplicity of data clouds. Once confronted with a multiplicity of data clouds, the users' task becomes one of establishing the meaning of the data grouped within the clouds. This should be easy. The user should just be able to point and click on an object and in viewing one or two objects, establish the nature of the data contained in any particular cloud. As the talk above (#10) makes available, 'just' pointing and clicking occasioned a great deal of work. The problem here is two-fold. On the one hand the point and click functions are difficult to use. Selecting a cloud (or 'hull') and selecting an object are controlled by the same device (the left click on an ordinary mouse) so the user all too often ends up selecting a cloud rather than object within it. On the other hand, and as pointed out above, the default weightings employed during the trail were problematic rendering cloud content meaningless. Both are fundamental to the use of the demonstrator from a users' point of view and both are easily remedied (weightings as discussed above and through the modification of mouse functionality). In the course of establishing the meaning of data, users also experienced some confusion when switching between 'locking' modes. Clearer indication of lock status is desirable here, perhaps through the use of colour: red for locked, green for unlocked, for example.

Browsing a multiplicity of data clouds. Users experienced no further troubles in browsing data clouds and worked on the basis of reading full titles of objects before moving on to other clouds.

Deleting data clouds from the Q-space. In the course of browsing the Q-space and finding particular data irrelevant to their search, users were introduced to the notion of deleting data clouds from the Q-space. This required some instruction from the designer and was readily understood such that searching subsequently proceeded through *refining the search* in browsing and eliminating data clouds.

Other issues. Users raised a number of practical concerns in discussion following the trial. Three issues are most pertinent to design:

1. The ability to manipulate the molecular chain of data clouds itself such that the chain could be turned around and its features viewed (a feature that could be assigned to the trackball).
2. The ability to read a list of objects contained in a cluster identified as interesting. This was considered a more effective way of viewing titles once the search has been narrowed down to a small number of objects.
3. The ability to issue searches using different search routines as available on the Internet.

Outcomes

In reading what might appear to be a litany of troubles and desires for improvement, it might be possible to draw the conclusion that much still needs to be done to make the library demonstrator practically useable. Any such conclusion may be attributed to the 'critical' stance of the evaluation strategy, which actively sets out to document practical troubles and identify areas for improvement as construed by end-users. While important to good design, this strategy requires balancing. The original user trial identified a number of areas requiring improvement. Consequently we have seen the construction of a reliable demonstrator that does not 'crash' when confronted with real world tasks. We have seen the 'scaling up' of the demonstrator to handle large searches. The demonstrator now furnishes users with full titles as resources for conducting their search activities. We have seen the implementation of labels informing users of the type contents of clouds. Users can now 'check' an object's classmark partners to see if similar objects satisfy their search requirements. More legible text displays have been provided along with a more flexible navigation device. Users can now pursue searching not only through browsing data clouds but also by a process of elimination, deleting irrelevant clouds from the space. Users can also retrace their search activities through the use of a history function. The achievements of the year's work are considerable then and consist of ostensive successes including the concrete specification of future work.

Future work

Several tasks remain to be accomplished from the original trial.

1. An efficient aesthetic layout of the visual field needs to be devised.

2. Q-pit based 'related keyword' search functions need to be implemented

3. As well as functionality enabling multiple searches to be related and queried.

These are currently under development. In addition, user trials of the new library demonstrator identify the following requirements.

1. The design and implementation of a single navigation device incorporating the functionality of a standard mouse with that of a 3D mouse.
2. Extension of the history function to support retrieval over several weeks.
3. Visible functionality for 'clearing' the Q-space.
4. Automatic zoom (in and out) that enables the user to view the entire space as the size of the Q-space dictates, rather than having to take manual action.
5. Modification of mouse functionality such that object and cloud selection is distinct and easy.
6. Colour indicators to inform the user of 'lock' status (on or off).
7. The ability to manipulate the molecular chain of data clouds itself such that the chain could be turned around and its features viewed (a feature that could be assigned to the trackball).
8. The ability to read a list of objects contained in a cluster identified as interesting. This was considered a more effective way of viewing titles once the search has been narrowed down to a small number of objects.
9. The ability to issue searches using different search routines as available on the Internet.
10. Perhaps the most significant challenge the demonstrator faces in the future is that of being scaled up to meet the demands of the Internet.

We consider potential solutions to these issues below.

1. An efficient aesthetic layout of the visual field needs to be devised

The interface to the Q-pit artefact has progressed in two directions, guided by a user interface designer and graphic artist. The first of these involves improvements to the Graphical User Interface controls and layout of the 'desktop' version of the search tool as used in the studies reported. The second direction involves the migration of the Q-pit to make it usable in the multi-user Deva environment displayed on the custom built panoramic navigator and cylindrical projection device. In this chapter, we concentrate mainly on the improvements to the desktop version. The development of an effective aesthetic layout is probably the major challenge left to be addressed and, accordingly, we devote some degree of time and space to the matter.

Towards an effective aesthetic layout: remarks on the Q-PIT interface. The main action performed with the Q-pit is navigation through the different data clusters, thus most of the screen space should be reserved for this. The navigation

is mainly carried out interactively without using interface elements. More sophisticated actions like going back and forth in the history, deletion of clusters or objects (etc.) is accomplished with buttons on the toolbar at the top of the navigation display. Next to these buttons is a field showing the current keyword. This field works like a pull down menu, displaying recent keywords plus the option to enter a new keyword. Less frequently used actions could go into several tabs at the bottom of the display. In navigation mode the tab section can be reduced to a small line showing only the tabs with their names. When one tab is clicked it will come up and cover a part of the navigation display. New users can easily create a new account. Every user has to log into the system to use his personalised bookmarks (etc.) and to send book requests to the library. The above issues sum up in three different elements of the interface:

- The button bar at the top of the display
- The navigation display in the middle
- A tab section at the bottom of the display

Below we consider each of these issues in more detail.

Button bar

The button bar consists of eight buttons and the field for keywords. The buttons are aligned in one horizontal line on the left of the screen. Below the buttons is the field for keywords. The button bar comprises the following controls:

- **New button:** clears the navigation display and the keyword field to start a new query
- **Back button:** lets the user go back to the previous visited object
- **Next button:** if the user already stepped back in history this button allows to go forward again
- **Stop button:** stops the movement of the clusters
- **Reshape button:** rearranges the clusters after stopping their movement (it combines the current “reshape” and “go” buttons)
- **Delete Button:** deletes the selected objects
- **Info button:** brings up the information tab showing information on the selected object.
- **Bookmark button:** adds the selected objects to the list of bookmarks
- **Navigate button:** removes the focus from the currently selected object and allows for free navigation
- **Exit cluster:** button lets the user go back from the selected cluster to the entire Q-PIT space (this button could be removed if moving out of a cluster was done by simple navigation).

- **Merge queries:** combines several queries that are selected in the queries tab (actually it allows to combine more than one keyword in a single query)
- **Request:** send a request to the library to reserve this book for the user
- **Keyword field:** text field with a pull down menu allowing to select one of the previous keywords and to enter a new keyword.

Navigation display

The navigation display shows the clusters that are part of the current query. If the cursor is over an object the name or description of the object is shown in an overlaid field at the mouse position. If the cursor moves away from the object the field disappears. The information that is shown in this field depends on the weighting of the search terms. If they are equally weighted it displays the title of an object (book title). The naming becomes more and more precise by approaching a specific object. Clusters are named with general names identifying the contained objects (e.g. the name of a section of the library). More comprehensive information on a selected object is available in the information tab, which can be invoked either by clicking the information button in the button bar or by clicking the information tab.

In the normal navigation mode the display shows only the clusters, objects, and trails. When an object is selected it is centred in the display and a semi-transparent white circle appears in the middle of the display. The selected object becomes the centre of rotation. Clicking the navigate button removes the circle and allows for free navigation. Selected objects are highlighted by a spinning box-outline. In the Q-pit it is possible to select several objects. Objects are deselected by clicking into free space.

The tab section

The tab section provides all controls that are not frequently used. The section normally is reduced to a single line showing just the different tabs. When a tab is brought up it is extended to its default minimum size to leave as much space as possible for the navigation display. The default minimum size is appropriate to display all contained sliders in the weighting tab or a reasonable amount of entries in the bookmarks list etc. Thus it is possible to navigate, for example, with the weighting tab visible. Tabs are opened by clicking on the name sign. They are closed by clicking the name sign again. Clicking on a second tab while one tab is already visible exchanges the two tabs. Dragging the name sign allows to extend or reduce the tab. The tab section contains the following tabs:

- **Weighting.** Shows seven sliders titled “ISBN”, “Title”, “Author”, “Publisher”, “Publication date”, “Publication Type” and “Classmark”. By means of these sliders it is possible to adjust the importance of these characteristics. The titles are placed next to the sliders.
- **Sliders.** Indicate the full percentage range (zero percent to the left of the slider and hundred percent to the right of the slider) and they show the actual percentage that is set (a highlighted left part of the slider from zero to the handle). The clusters are re-arranged according to the weighting in real time.
- **Information**. Shows information on the selected object. If the selected object is a cluster a general characterisation of the contained objects is displayed. If the selected object is a book its cover image and some descriptive text is displayed.
- **Bookmarks**. Shows a list of all objects that have been bookmarked throughout different queries (the bookmarks are personalised). Selected bookmarks can be removed from the list by clicking the delete button in the button bar. The bookmarks show the same characteristics as the sliders in the weighting tab. They can be rearranged according to the different characteristics.
- **History.** Shows a list of all visited objects within one query. The tab is organised in the same way as the bookmarks tab. Going back and forth in the history can either be done by clicking the “back” and “next” button in the button bar or by selecting one of the objects in the history tab. The currently centred object is highlighted in the list. When objects are deleted in the Q-pit space they are automatically greyed out in the history list. Clicking on one of these objects prompts the user if he wants to bring the deleted object back into the query. The objects in the history list are organised in a loose chronological order.
- **Chronological Order.** This mean that going back and starting a new navigation path does not remove the previous path from the list but adds the new objects. Objects can be deleted from the history list. The path that is visible in the Q-pit space is then adjusted.

- **Queries**. Show a list of all previous queries. This list is also personalised so that the users can also find the queries they made in other sessions. These are identified by the keyword in combination with the weighting of the different characteristics. In the queries list it is possible to select several queries and to merge them by clicking the "merge" button in the button bar. This allows users to combine more than one keyword in a single query. The combination of several queries creates a new query in the list. Every query is stored in its last state before a new query was started. It is stored with its history so that the user can retrace the development of the query.
- **Account.** Shows an overview over the personal account (borrowed books, requested books etc.).

These features are presented to user through an interface designed in conjunction with ZKM and presented in the following chapter. The interface provides access these features through a context driven interface.

2. Q-pit based 'related keyword' search functions

In the Sienna Demonstrator, the so-called "trails" database was visualised using the *10_dencies* art piece. It was intended to replace this piece with a further Q-space, using Q-pit itself to display the graph of keyword relationships.

By returning to the data originally used to generate the 10_dencies art piece, we discover the following two relations.

trail						
field	**trailid**	**Type**	**termref**	**dtime**	**ip**	**isbn**
type	int	Char	int	datetime	char(15)	char(10)

searchterms		
field	**termid**	**Term**
type	int	Varchar(255)

Fig. 1: The *trail* and *searchterm* tables of the *library* database. The *termref* field in table *trail* references a unique entry in the *searchterm* table.

By searching the "searchterms" table, we can convert the current search term into an underlying "termid". We can then in turn search the "trail" table and retrieve a set of tuples that represent the trails (the sequence of search terms surrounding the current term) that the current term participates within. Where this is more than one trail, we can generate a graph similar to the one shown below in two-dimensions.

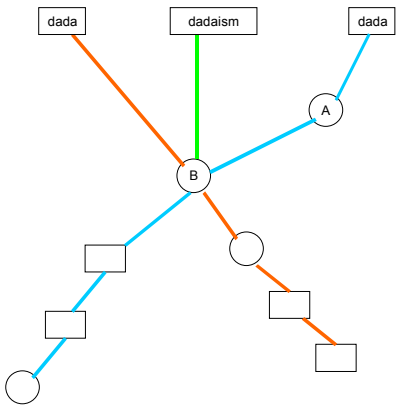


Figure 1: A 2-D presentation of an object 'B's trails

Clearly, this is just another graph and it would be possible to run this through the Q-Pit force displacement algorithm to generate a new Q-space which would then represent the set of related keywords to the current search term.

We are currently engaged in implementing this functionality as part of the Q-Pit engine. Care will have to be taken when representing this information as part of the visualiser, allowing the user to easily distinguish between a Q-space of books and an associated Q-space of related search terms. The interface will have to be refined to allow the user to select a new search term from the search term space, in order to generate a new Q-space of books.

3. Enabling multiple searches to be related and queried

The combining of the semantics of a number of searches is in principle a straightforward operation, involving the merging of two of the tuple-spaces and the recalculation of the minimal spanning tree and similarity matrix on the aggregated space. One advantage of the techniques used in the Q-pit layout is that once similarity calculation has been applied, the remaining stages of the process work on the unitless 'similarity' value rather than relying on any particular field of the original data. Thus, if appropriate similarity comparison functions can be determined, any form of data may be compared with any other, and such operations as merges/unions and intersections become trivial. The difficulty with the merging multiple searches, then, is not one of algorithmic or architectural complexity, but rather of presenting a comprehensible mechanism via the user interface. The proposed 'tab' set of the new graphical user interface layout in the previous section goes some way we believe towards remedying this.

4. Alternative navigation mechanism

There is no general easy solution to the navigation of a true three dimensional space. The 'default' navigator applied by MAVERIK to its environments proved to be entirely inappropriate for all but the most practised of users. The trackball and mouse solution used in the user trials demonstrated a significant

improvement, but was not without its problems (not least the confusion caused by having two ‘mouselike’ devices close together). Here we present a proposed navigation routine based on the semantics of the Q-pit that may prove appropriate in addition to potential hardware solutions to the problem.

Customised 2D mouse navigation of the Q-pit

The navigation in the Q-pit can be done with a two-button mouse. By simply moving the mouse it is possible to examine the currently visible space. ‘Mouse over’ events are detected and objects can be selected by clicking on them. Dragging with the left mouse button pressed shifts the view. Dragging down moves the view down, dragging up moves the view up, dragging right moves the view to the right and dragging left moves the view left. Dragging with the right mouse button pressed rotates the view. Dragging down rotates the view downwards, dragging up rotates the view upwards, dragging right rotates the view to the right and dragging left rotates the view to the left. Dragging with both buttons pressed zooms the view. Dragging up zooms in and dragging down zooms out.

An alternate device

The user trials suggest that use of one single device for all interactions is easier to handle than using a trackball for navigation and a separate mouse for selection and interaction with the interface widgets. Aside from the software navigator solution proposed above, another potential way around the difficulties of having two input devices would be to combine in a new peripheral the functionality of the trackerball, and that of a standard mouse. This device could be similar in design to a ‘wheelymouse’ where the third central button is a rotating disk for controlling scrollbars, however in this ‘navi-mouse’ the central button would be replaced by a small trackerball to control orientation in the three dimensional space, with a thumbwheel on the side for zooming in and out.

5. Extension of the history function

So far, the demonstrators have been oriented to a single session. The session begins with no prior knowledge or information. Once the session is over, no results are saved. Quite apart from the fact that support for multiple sessions is a good idea, this demand shows that users approach information retrieval systems with certain expectations, and this now includes the concept of ‘history’. It would seem to be relatively straightforward to extend the demonstrator’s database support to include a logging mechanism that would keep session history. At the very minimum, it would be possible to store (user, keyword) pairs that would allow the user to regenerate the spaces they had previously explored. At the other end of the spectrum, we could keep details of the books they visited and thus

recreate the illuminated ‘search path’ through the data, and also details of the clouds they deleted. At any rate, the system extensions needed to support a multi-session history function appear possible and is being considered in the design of the new graphical user interface for the device.

6. Visible functionality for ‘clearing’ the Q-space

At the simplest level, this means we need a button marked ‘clear’. Although it is possible for users to select clouds and portions of the Q-space and to delete them piecemeal, it is clear that users expect and require a simpler mechanism for deleting the results of an entire search. The ramifications of this are a need to identify and select the distinct spaces generated as a result of specific searches, making them the target of the ‘clear’ button action.

7. Automatic zoom

This is the return of the ‘centred view’ requirement from the previous study. The original problem was that the space was generated and the user maintained their current ‘random’ position in the space, and had to manually navigate to a more centred position. While the current demonstrator correctly positions the user in the centre of the space being generated, it does not ensure that all the objects in the space are visible. The user can manually zoom out (or in, if all objects are visible but quite far away) to centre the view, but this requirement states that we should do this automatically. Since MAVERIK by default maintains a structure containing all the bounding boxes of all objects, composite and otherwise, manipulating the viewpoint so as to automatically fit all objects is a relatively trivial task.

8. Modification of mouse functionality

The problems of selecting an object within a cloud return to haunt us. In the previous implementation, a bug caused erratic selection of objects and hulls. In the current version, an ill-considered policy decision as to when to treat hulls as being selectable and when to ‘select through’ them to their contents caused some difficulties for users during the trial (the existing policy was to treat the first hull encountered as being ‘select through’ as long as a target object was found as the next closest object; for the majority of purposes this is the expected result). However, attempting to select an object in the far distance through several layers of hulls yield inappropriate results. The policy is trivially fixed. In addition, an experimental selection mechanism has been implemented whereby the left mouse button selects objects (only) and the right mouse button selects hulls (only). Whether this or the technique described in the section on interface layout proves more successful remains to be seen.

9. 'Lock' status

Often users failed to spot that their navigation was still 'locked' around an object. Though the navigation icon in the centre of the display indicates 'locked' (a circle) or 'free' (arrows pointing away from the centre), this was not clear enough. Two features have been implemented to improve this; firstly a change of colour from white to green when the lock is engaged or broken, and secondly an automatic 'breaking' of the lock when a user moves far from an object (since users distancing themselves from the selected object was usually an attempt to break free in order to investigate another area of space or create a new query).

10. Manipulation of the molecular chain

This feature was partially implemented by selecting an object and 'locking' navigation around it. However users requested the ability to rotate an object 'at a distance'. This feature could easily be associated with the selection of a query from the new interface layout, and the rotation/inspection of the query at a distance associated with movement of the trackball.

11. Viewing a list of titles in a cluster

This raises issues of when does the 3D representation of the books cease to offer any advantage over a 2D textual presentation. It is clear that the 3D representation is useful where there are hundreds or even thousands of books or items in the space, and the users can quite happily interact with the space to generate and investigate clusters of objects. With the new facility to delete objects and clouds deemed unworthy of further interest, users can pare down the space towards a minimum. They will then clearly reach a point where a straightforward scrollable text list of remaining items will meet their needs. This is clearly achievable within the desktop implementation of the system.

12. Issuing searches using different search routines

Again, this requirement is a measure of user sophistication and expectation due to exposure to the Internet and to the search engine facilities therein. What this requirement actually refers to is the ability to construct more complex Boolean expressions in the formation of a selection condition. In our current demonstrator, we are limited to the facilities offered by the underlying WebOPAC system, and thus we do not currently offer such complex facilities. Having said that, however, it would be possible to build a more complex layer over the fundamental WebOPAC facilities and implement search routines ourselves within the layer. This appears possible and also worthwhile.

13. Scaling up

One should imagine that it is possible to develop the Library Demonstrator so that it is connected to multiple libraries across the net (using the Z39.50 protocol for example) and thus retrieving a set of books from more than one source results in M (the number of books in that keyword) * N (the number of network sources), so that we could be dealing with many thousands of books. The concern here is, as with the Sienna Demonstrator, could we again scale up by an order of magnitude? This, as before, has impact on the order of the algorithms we use to generate the space. It has an impact on the visual representation we create: does a space with hundreds of thousands of books retain any meaning? The way in which the representation is created may also come into question: is there some way in which we can create and display clouds independently of the books that make up those clouds? Could the initial shapes of the clouds reflect the size and nature of the content of the cloud in a generic way rather than the individual books? Would it then be possible, with the passage of time, to populate the space with books in the background or on demand? Each cloud itself might represent a further Q-space. Here, we are invoking the concept of *containment*, a popular technique in computing for dealing with large numbers of objects. It may be that it is only with the challenge of size that the true power, if it exists, of the cloud concept would come into being.

In this chapter we have reported on the calibration of the developed Q-space and outlined the issues to be addressed by the evolution and amendment of the Q-space. The amended Q-space was assessed in conjunction with a number of real world users and the identified issues were addressed through the amendment of the prototype and provided input for the redesign of the interface presented in the next chapter.

The focus of this year's work has been to respond to the end-user trials carried out on the Sienna demonstrator. To this end, we have concentrated on improving the reliability of the visualiser. This has allowed us to improve on aspects of the visualisation, including the amount of data we are able to visualise, the basic speed and responsiveness of the visualisation. We have therefore been able to generate and deform dataclouds in real-time, providing the users with the sense of greater control over the space they are exploring.

Ultimately, in addition to the positive response to changes by end-users, this year’s work is marked by a call from users for extended trials. That is, for trials in actual practice. This marks a considerable success for the eSCAPE Project and these trials have been used to inform the development of a design-oriented interface. This design-oriented interface also allows this electronic landscape to be integrated into the Placeworld demonstrator described in part I of this deliverable. The following chapter considers the design of this abstract escape interface.

Chapter 9: The Design of the Abstract eSCAPE Interface

Andreas Kratky
ZKM | Zentrum für Kunst und Medientechnologie Karslruhe

The Q-PIT exists in two different versions, the abstract eSCAPE using a touch sensitive screen for navigation and the desktop version running on a normal desktop computer. The different input devices also require two different interface concepts. The panoramic navigator of the abstract eSCAPE provides a handle for navigation by e.g. turning the entire navigator and the possibility of touching/clicking on the touch sensitive screen. Thus the interface has to be designed in a way that all actions can be done only by clicking. All buttons have to have an appropriate size for handling with the fingertips. The widgets are designed as semi-transparent overlays that are only visible when needed in order to keep the screen as clear as possible for the actual dataclouds.

The desktop version will run on a normal computer either equipped with a normal mouse or with a specially manufactured mouse with an integrated trackball. Navigation within the dataclouds is carried out with the trackball or mouse button combinations and for selection etc. the mouse buttons are used. We will look at navigation in more detail in a later section.

In the normal navigation mode only a small oval at the bottom of the screen is visible giving access to the main section of the menu:

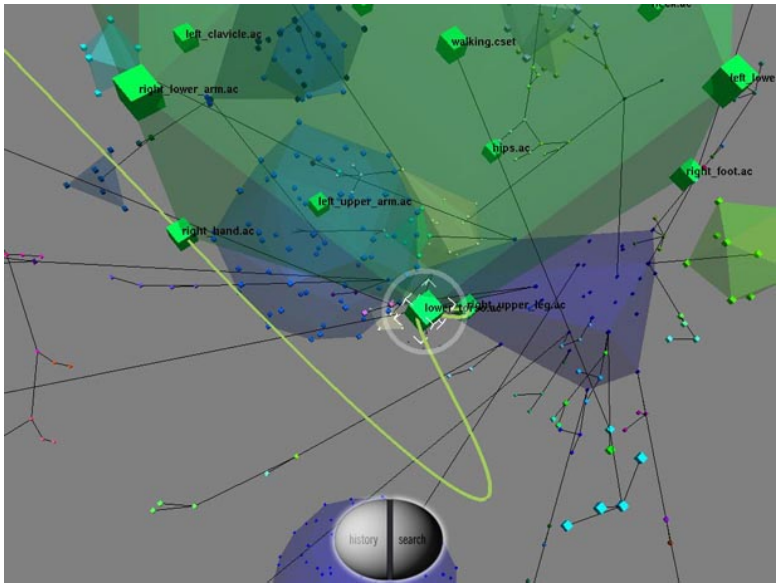


Figure 1: Normal navigation mode

The main menu is visible as an oval shape centred at the bottom of the screen. The oval is divided into two parts one giving access to the history submenu and the other giving access to the search submenu. Clicking on one of these parts brings up the corresponding submenu.

History Menu

The history menu provides the options “back” and “next”. Clicking one of the options removes the submenu from the screen and executes the action.

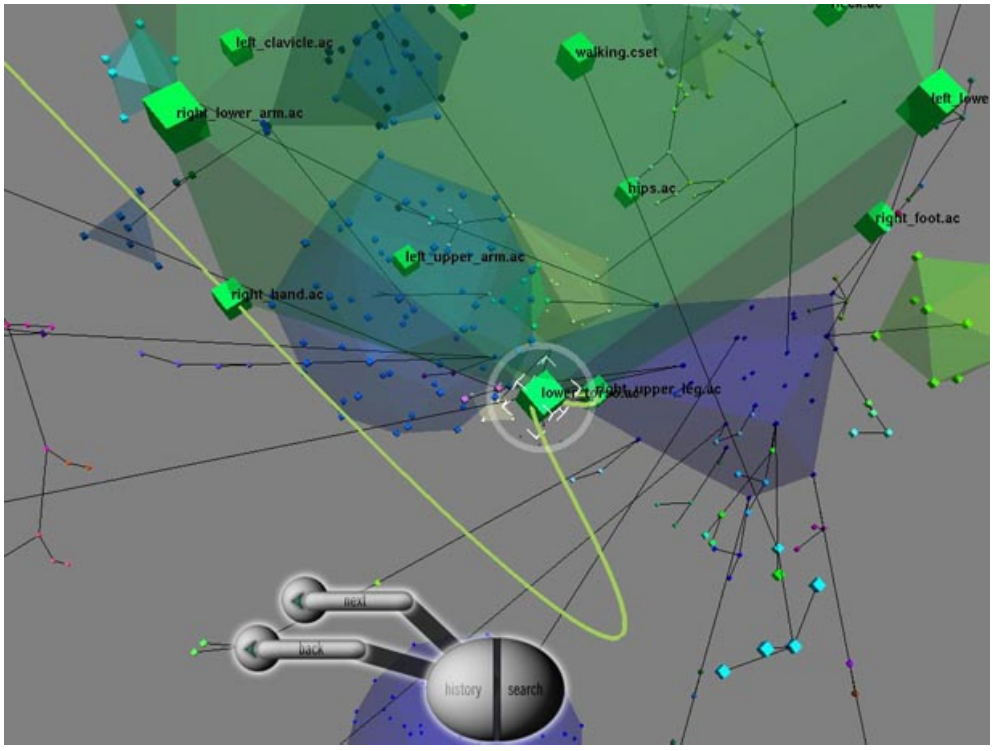


Figure 2: The history section

Search Menu

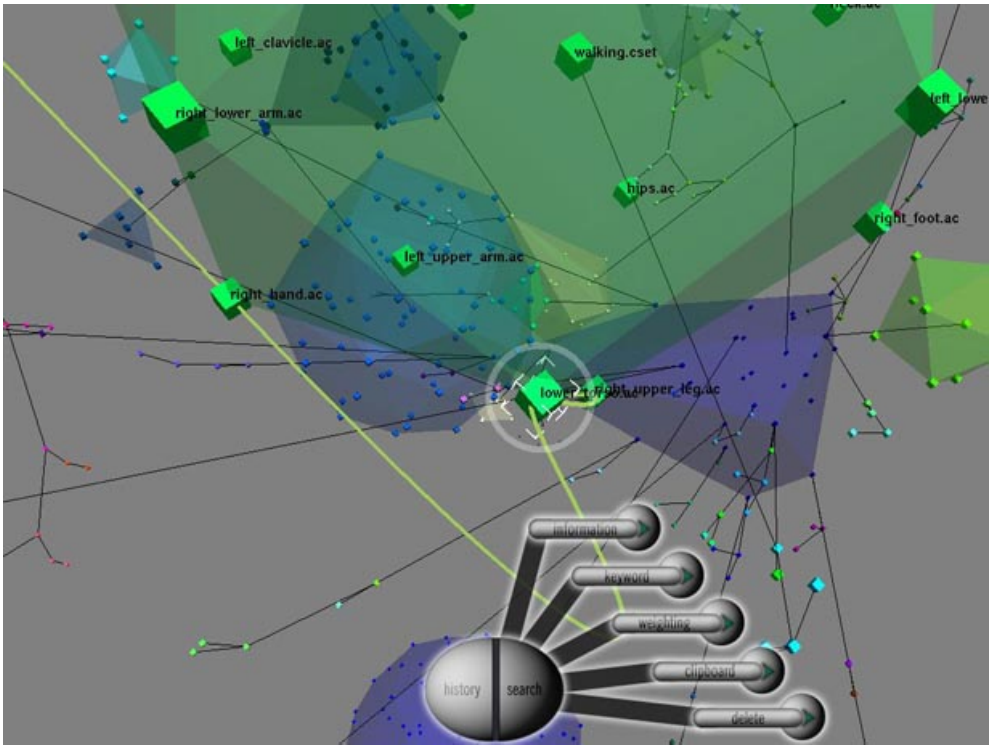


Figure 3: The search section

The search menu provides the options “information”, “keyword”, “weighting”, “clipboard” and “delete”. The options “information” and “delete” are only available when an object is selected otherwise they are greyed out.

Information option

Selecting an object and then clicking “information” brings up the information-window. The information window shows the cover image (if the selected item is a book) and descriptive text on the selected object.

It provides a button that allows users to add the selected object to the clipboard and a “cancel” button to close the window. If dragging is possible with the touch screen the window can be displaced by dragging its title bar.

left_clavicle.ac
walking.cset
right_lower_arm.ac
information
Professinal WAP
The Wireless Application Protocol (WAP) provides a universal open standard for bringing Internet content and advanced services to mobile phones and other wireless devices. Designed from the start to take account of the limitations of today's devices - small screen, low power and low bandwidth - WAP is the platform for the new generation of "media phones".
Professional WAP gives a broad overview of WAP and associated technologies - what it is, why it's useful and how to take the best advantage of it. It covers the WAP architecture and protocols, what WAP tools and servers are available for developers today and how to WAP-enable your web server.
Practical examples will show you how to make content and services available over WAP using WML Decks and WMLScript. More advanced topics include administration, transactions, security, and integration of telephony with internet services (WTA).
The book is principally aimed at web developers who are experienced in HTML, scripting, probably some server-side technology, and who now want to find out how to apply their skills in the WAP world. It will also appeal to project managers, application architects or other professionals who want to gain an understanding of WAP technologies and how businesses could benefit from them.
cancel
add to clipboard
right_foot.ac
history
search

Figure 4: Showing information about the book

Keyword option

Clicking the “keyword” option brings up a window showing a keyboard. On top of the keyboard is a field that shows the current keyword. The field provides a pull down menu which is invoked by clicking the arrow key to the right of the field to access previous keywords. The keywords are displayed in a text field on top of the keyboard window. This window provides a scroll by line mechanism if needed. The window is closed either by selecting one of the keywords or by clicking somewhere outside this window.

The current keyword is automatically selected and can directly be overwritten. The keyboard works as a normal keyboard and allows the user to type in a new keyword.

The window has an “ok” button to close the window and confirm the changes made to the keyword and a “cancel” button to close the window without changes. If dragging is possible with the touch screen the window can be displaced by dragging its title bar.

Figure 5: Keyword Entry

Weighting option

Clicking the "weighting" option brings up a window containing three sliders to specify the weighting of the search terms. One slider is related to the author, one to the title and one to the date. Clicking the plus or minus buttons at the two ends of the sliders increases or decreases the value of a slider.

The window provides an "ok" button to close the window and confirm the changes made to the weighting and a "cancel" button to close the window without changes. If dragging is possible with the touch screen the window can be displaced by dragging its title bar.

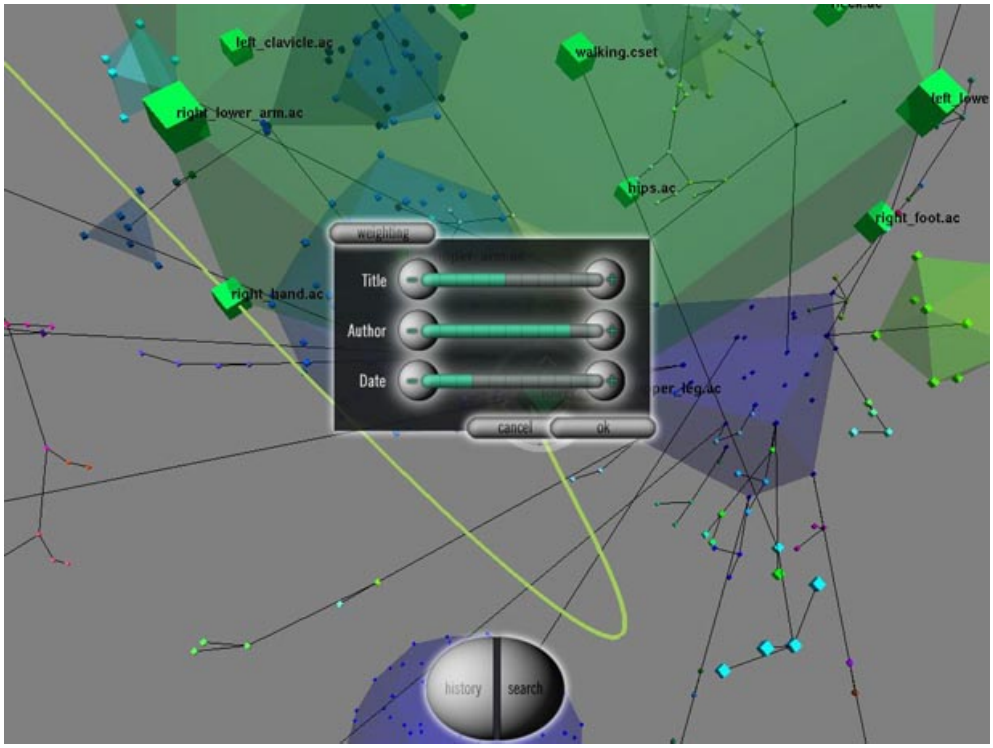


Figure 6: Specifying weightings

Clipboard option

Clicking the clipboard option brings up a window showing objects that were copied to the clipboard as bookmarks to place them somewhere else in the Placeworld or just to memorise specific objects. With the clipboard it is possible to carry objects to different places. The objects are visualised by icons showing either a box for a single book, a polygon for a hull or any other representation for objects that can be collected in the Placeworld. The icons are organised in rows that can be browsed by means of a scrollbar. Objects in the clipboard can be selected by clicking on them. They are then highlighted and every action such as "place" or "delete" refers to the selected object. Multiple selection is not possible with the panoramic navigator.

The scrollbar provides two arrow keys and a scroll handle. Clicking one of the arrow keys will scroll the objects line by line. The scroll handle is mainly to indicate the current location in the window. If dragging is possible with the touch screen, scrolling can also be done by dragging the handle. Also, in this case, scrolling works line-wise.

The window provides:

- an “add selected” button to add the object that is selected in the Q-PIT data cloud; the button is only available when an object is selected in the Q-PIT data cloud; the added object is automatically the active one;
- a “delete” button to remove the selected clipboard-item from the clipboard; the button is only available when an object is selected in the clipboard;
- a “rename” button that invokes a keyboard window to enter a new name for the selected clipboard-item; the button is only available when an object is selected in the clipboard; the keyboard is displayed on top of the clipboard window;
- a “place” button to place the selected clipboard-item in the Placeworld; the button is only available when an object is selected in the clipboard;
- a “cancel” button to close the window.

Figure 7: Showing the clipboard

Delete option

Clicking the delete removes the selected object from the Q-PIT data cloud. The button is only available when an object is selected.

Navigation in the Q-PIT

Startup

After starting the Q-PIT an initial cloud constellation is visible showing the general sections of the library with names identifying these sections. The weighting of the keywords is set to zero for all search terms.

Entering a keyword

The clusters are arranged corresponding to the keyword. They are named with a more or less general identifier appropriate to the keyword. Any new keyword results in a new constellation of clusters. Keywords are entered by clicking “search / keyword” and typing a keyword.

Setting the weighting of the search terms

The constellation of clusters can be influenced by giving either the “author”, the “title” or the “date” a higher importance. This is done by clicking “search / weighting” and adjusting the sliders in the appropriate way. The clusters are reshaped in real time.

Selection of a cluster

By navigating through the Q-PIT space it is possible to examine the different clusters and dive into them to see their contents. The clusters have semitransparent hulls of a specific colour. By approaching a cluster the contained objects become visible. Diving into a cluster makes the hull more or less opaque obscuring the objects outside the cluster. Like this the user can concentrate on the objects of interest but does not loose the relation to the whole Q-PIT space. By navigating inside the hull it is possible to examine the contained objects. If the internal objects of the cluster which are themselves organised in clusters they can be examined in the described way.

Navigating out of the hull lets the user navigate in the Q-PIT space.

Navigation in space is done with the panoramic navigator. The centre of rotation is always the centre of the selected item. If no item is selected it is the centre of the whole query (i.e. the centre of all clusters) in which the user is currently navigating.

Chapter 10: Developing a Generic Architecture

Peter Phillips
Lancaster University

The abstract landscape demonstrator developed last year and amended as described in Chapter 6 is a system to automatically generate abstract landscapes, known as Q-Spaces, based on the results from searches sent to the Lancaster University library's Online Public Access Catalogue (OPAC). These Q-Spaces are generated by a Q-PIT system modified to create clusters, or groups, of similar objects. The visualisation presents these interfaces in 3 dimensional space. While this approach meets the immediate needs of the user of the library OPAC system. In these versions of the Abstract Electronic Landscape Demonstrator, the generation and presentation of an individual Q-space (a 3 dimensional representation of a result of a search upon a Library book catalogue) are tightly coupled. In this chapter we describe the development of a more generic service that decouples presentation from data generation. This generic service allows the Q-space to exploit a broad range of information sources and to present this to a large number of interfaces of different styles. The diversity of these interfaces is reflected in the alternative interfaces presented in chapter 11.

This chapter will outline the architecture and use of the previously reported version of the abstract landscape demonstrator and review the results of the user evaluation of the system together with requirements drawn in from related areas of work within the project. It will then focus on the new architecture and its use, before concluding with a discussion of the benefits that have been gained by introducing these changes.

The Existing Library Demonstrator Architecture

The abstract landscape demonstrator offers users a novel way of interacting with results obtained from a keyword search on a University library catalogue, known as OPAC. The traditional methods of accessing the OPAC system are via a telnet session or web-based form. In both cases a user invokes a search of the books in the library catalogue by submitting a keyword, a word somehow connected to the book(s) that they are seeking. The OPAC system lists the books that it has found in its search as a series of telnet screens or web pages. One of the motivations for the abstract landscape demonstrator is to somehow represent all the results of the search at once, rather than as pages, and show the similarity of one item (usually a book, but not always) to another. This is aimed at supporting 'fuzzy searches', where a user who does not know the structure of the library can find the books

that they want, and the opportunistic discovery of books that meet the user's needs but that they might not have considered using a normal, page-based interface. The discovery of the 'existence of unknown-but-possibly-suitable' search items [Crabtree] is best suited to situations where the complete set of search items can be visualised at once, which is why the demonstrator uses 3 dimensional space to display the results of the search.

The search results are placed into 3D space and, in addition, similarity and grouping features are calculated, a process that is not performed by the OPAC system. This extra processing, performed by the Q-PIT processor [Mariani], aims to inject the 3D representation with coherence and legibility, particularly in the application of grouping (also know as 'region') features to the displayed search results. The 3D representation is known as a Q-Space and is best described by drawing a comparison with a classic molecular model. Each atom in the Q-Space molecule represents a singular search item. The bonds between each atom are 'bonds of similarity'. The addition of the grouping data manifests itself as a cloud enclosing all atoms that are calculated to be members of that group. Figure 1 shows a typical Q-Space where a label has been added to each search item in the space to aid identification.

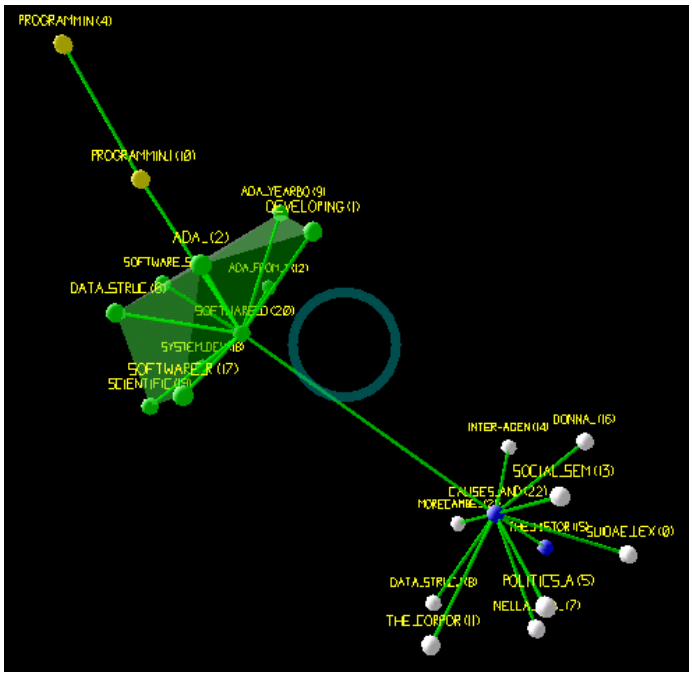


Figure 1: A single Q-Space

Components of the Year 2 Abstract Landscape Demonstrator Architecture

OPAC : On-line Public Access Catalogue

The OPAC system is the online catalogue of Lancaster University Library. It is accessible via telnet and the World Wide Web. Due to the nature of its implementation it was decided to present this service as a single shared repository for further components of the architecture to interact with.

Data Extractor

This piece of software acts as a bridge between the OPAC system and the remaining architecture. When it receives a keyword search from an end-user it checks a 'cache' of previously searched keywords to ascertain if the results are already contained in the shared repository. If the results are found in the repository the Data Extractor returns a query identifier associated with these results. If the results are not found then the Data Extractor forwards the keyword to the OPAC system. The results returned by OPAC are loaded into the shared repository and the associated query identifier for these results is returned to the end-user. The keyword is also saved to indicate the presence of its associated results in the repository. Once this process has completed the Data Extractor takes no further part in the (re)generation and management of a Q-Space.

MySQL Database and JDBC

The single repository of keywords and associated results is implemented by using a mySQL [mySQL] relational database. A JDBC [JDBC] interface library is employed to allow shared access to the repository from different components of the architecture.

Q-PIT and DIVE

The processing of search results that are returned in response to an end-user's keyword takes place in a Q-PIT, modified to support the calculation and production of data clouds (the grouping or region features of a Q-Space). The algorithms used to generate the Q-Space from these results are explained in detail in [Mariani and Colebourne], but a brief summary of steps required has been included below.

1. Generate a Benediktine space [Benedikt] for initial "random" positioning of points within the space
2. Generate a similarity matrix
3. Generate the minimal spanning tree
4. Identify regions within the tree
5. Apply a force displacement algorithm
6. Generate the data clouds

Further explanation of each of these steps is included later in this chapter.

The end-user's visualisation of the results from their search takes place in 3 dimensions via the DIVE platform. The demonstrator tries to display the Q-Space dynamically as it is being generated; the DIVE interface shows an animated display of the processing taking place in Q-PIT. DIVE is also responsible for handling end-user interactions with the Q-Space itself, once it has been generated. The primary method of interaction with the Q-Space is through a Head Up Display (HUD) and 'vehicle' (a method of navigating through the space) developed specifically for this type of Q-Space. The vehicle has two modes of interaction, 'point-and-click' and 'browse'. The first mode of interaction allows the end-user to select individual search items from the Q-Space molecule. The item's details are displayed in a section of the HUD, but can also be viewed in an external Java window. The second mode of interaction, the 'browse' mode, fixes the end-users gaze on a search item that they have previously selected and allows their viewpoint to orbit around the item. This mode was designed to allow exploration of the Q-Space but still retain a sense of position within it. It would be easy to become disorientated in a Q-Space using traditional 3D navigation techniques, such as forward, back, up or down – particularly as the Q-Space is by definition an abstract space. There is no up or down and the only landmarks that are available are the visualisation of search items themselves.

The visualisation and collection of end-user parameters, which affect the generation of a Q-Space, is handled by a GUI interface under the direct control of Q-PIT, rather than DIVE. Some of the processing steps within the Q-PIT are executed again during the lifetime of a Q-Space in order to regenerate the space based on parameter values selected by the end-user. The high degree of interactivity required by the demonstrator means that, in this architecture, Q-PIT and DIVE are very tightly bound together. The Q-PIT, written in Java, has to access DIVE's internal representation of the Q-Space to be able to add, remove and update parts of the Q-Space as quickly as possible. To this end a Java interface to the DIVE system has been developed [Stenius and Trevor], allowing Java applications access to DIVE maintained objects.

The Architecture as a Whole

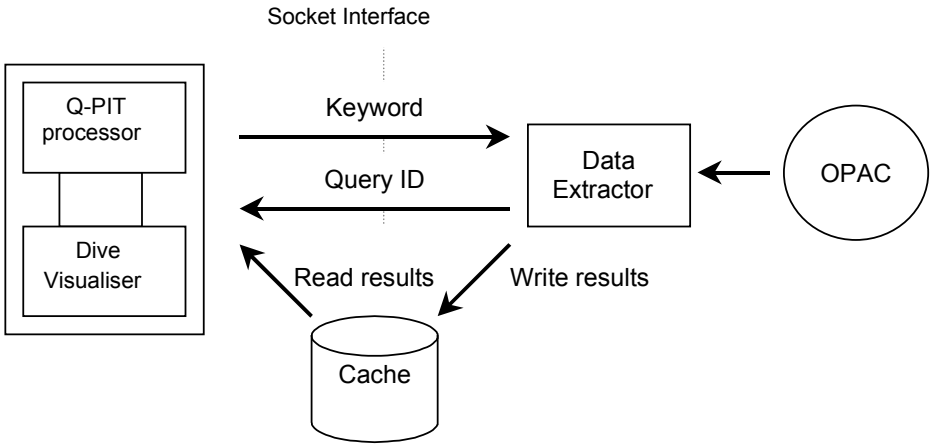


Figure 2: The Complete Architecture

Figure 2 shows the placement of the architectural components, outlined above, in the overall demonstrator structure. In the next section we review the process of generating a Q-Space and show the sequence of events that take place within and between components.

Generating the Q-Space

The whole process of generating a Q-Space is invoked by the end-user issuing a request to the Library visualiser (Q-PIT/DIVE). This request takes the form of a keyword to be used in a search of the library catalogue (OPAC). In this section we trace how a Q-Space is generated and returned to the end-user once they have issued their keyword.

Populating the repository

The end-user's keyword is passed from the visualiser (Q-PIT/DIVE) to the Data Extractor via a socket interface. The Data Extractor, which communicates with the mySQL repository and the OPAC system, checks to see if the keyword has been previously processed and the results stored ('cached') in the shared repository. If the results for a particular keyword are found to be in the repository (known as a cache 'hit'), the Data Extractor returns a query identifier to the visualiser. If the results for the passed keyword are not found in the repository (a 'miss'), the Data Extractor must query the OPAC system itself via a socket connection. The keyword is passed onto OPAC and the returned results extracted using a simple text-based protocol. The Data Extractor has to parse these text results, extracting each search item in turn and converting it into a record that can be stored in the repository. These records are added to the repository though the use of JDBC

technology. Once all the search items have been parsed and stored the query identifier for the new set of records is returned to the Q-PIT/DIVE visualiser.

The repository itself, implemented as a mySQL database, has a schema that matches the schema of the data being parsed from the OPAC system. For each record that is returned, there are seven attributes:

1. 'ISBN': ISBN
2. 'Author': Author
3. 'Title': Title
4. 'Pub': Publisher
5. 'Pubtype': Publication type (a Book, CD-ROM, oversized pamphlet)
6. 'Pubdate': Publication date (a year)
7. 'Classmark': Classmark (a topic and location identifier specific to the library at Lancaster)

Producing the Q-Space

Having received the end-user's keyword and passed it on to the Data Extractor, the Q-PIT/DIVE visualiser receives a query identifier in reply. The Q-PIT component uses this identifier to form a query for the mySQL repository, which will return the list of search items found for that query. The execution of the query and access to the results is handled by a JDBC interface to the repository.

Once the results have been received from the repository, the Q-PIT can begin generating the Q-Space that represents the search. This process, which involves the six steps that we have already touched on above, is quite an intensive process. One of the features of the demonstrator is the way in which the end-user sees the production of the Q-Space and an animation of the search items that constitute it. Such feedback to the user is only possible because of the intimate access that the Q-PIT code has to the DIVE system. Here is a more detailed look at the six steps that the Q-PIT needs to take to generate a Q-Space (More details can be found in [Mariani and Colebourne]).

Generation of a Benediktine space

This process maps N-dimensional data (in this case N = 7, the number of attributes contained in each search item) onto a 3-dimensional display. Three of the attributes are mapped to 'position in space' (x,y,z), also known as the 'extrinsic dimensions', whilst the remaining attributes are mapped to an object's appearance, or 'intrinsic dimensions'. This is achieved by creating an ordered list of 'domain values' for each attribute in the repository schema. The choice of which three attributes to map onto numerical values, to be used for positioning of a search item, and which attributes to map onto appearance or behaviour is made in a mapping file. This mapping file is a simple text file that contains a list of the seven attribute names in an OPAC search item and whether they should map onto extrinsic or intrinsic dimensions.

Once the domains have been calculated, the 3D representation of the search items (the ‘atoms’ of the Q-Space) are created in the DIVE visualiser. Initially they enter the 3D world at the centre point of the Benediktine space but slowly begin to move to the Benediktine positions as an indication of the progress of the similarity matrix calculation.

Generation of a Similarity Matrix

This is collection of multiple calculations where every record has to be compared with every other record and a numerical value generated for each comparison and stored within the similarity matrix. A single similarity calculation involves two records. Within that calculation corresponding attributes from each record have to be compared. There are seven attributes in an OPAC based record, so a single similarity calculation will involve seven attribute-attribute similarity calculations. The attribute similarity calculation is based on the separation across the attribute domain between the two values being compared. The result of the single similarity calculation is the average of all seven attributes' similarity calculations. The demonstrator allows some control over the final result of a similarity calculation through the use of user-modifiable weighting parameters. The use of these parameters will be discussed below, but they are worth mentioning here as this is the point in the processing where their influence is brought to bear on the generation of the Q-space. From this point onwards, the calculations being performed go beyond the classic Benediktine space on which Q-PIT was founded; these calculations are aimed at adding legibility to the generated Q-Space.

Generation of a Minimum Spanning Tree

This processing step adds links between search items (the ‘similarity bonds’ between the ‘atoms’) by converting the entire similarity matrix into an undirected graph and using an implementation of Kruskal’s algorithm [Weiss] to reduce the graph to a tree. Each node in the graph represents a search item, each edge in the graph represents a similarity between two search items and the weighting of that edge is the actual similarity value. End-user notification of the progress of the minimum spanning tree calculation takes the form of a progress bar in the DIVE visualisation. Once the tree has been found, the edges that make up the tree are added to the visualisation.

Identify regions within the tree

Comparing the ratio of an edge with the average of its neighbour edges identifies regions within the tree. If the result falls below a certain threshold the edge is eliminated from the tree. Once this has been repeated for each edge in the tree, the resultant sub-trees are deemed to be regions of similar items. The colouring of search items in the DIVE visualiser indicates the progress of this calculation. Items sharing the same colour are in the same region (group).

Applying a force displacement algorithm

The search items within the Q-Space are now moved from their Benediktine positions to a position calculated by force displacement. The idea behind this algorithm is that search items will repel each other and search items connected by a common edge will attract; the calculation is said to have finished when the entire representation reaches some sort of steady state. The application of the forced displacement algorithm is completely animated, so the end-user can see how the forces effect the entire Q-Space and how it eventually settles down to its final position. As the animation progresses the edges between search items move with the items themselves, producing a smoother, more natural animation.

Generation of data clouds

Based on the region information calculated earlier and knowing the final position of the Q-Space molecule, the clouds that represent groups or clusters of data can be added to the visualisation. In the DIVE visualisation a cloud is simply a hull that encloses all the objects that are members of the group that it represents.

Once the six steps outlined above have been completed the user can interact and modify the way that the Q-Space is recreated and represented. Some of these interactions are highlighted below.

Further interactions

Once the initial Q-Space has been automatically generated the end-user has the opportunity to modify the parameters on which Q-Space calculations are based. Any changes that they make to the parameters are fed back into the Q-PIT processes and the space is regenerated. Wherever possible internal Q-PIT values, such as the Benediktine domain lists, are not recalculated, because the set of results being manipulated has not changed.

Factors: Weighting and Exactness

In the earlier description of the similarity matrix calculation it was pointed out that there was an opportunity for the user to influence how the similarity value for a comparison between two search items could be influenced. This influence manifests itself through the use of two factor parameters, weighting and exactness, which change the emphasis and precision of the similarity calculations.

The weighting factor is used to increase or decrease an attribute-attribute comparison's overall effect on the final aggregate similarity value. If the end-user wants to remove an attribute comparison from the similarity calculation, they would set the weighting for that attribute to zero. If they wanted to magnify the effect that an attribute comparison will have, they would set the weight to a number higher than one. Positive decimal numbers are valid weighting values. This also means that an attribute comparison can be 'turned down' by giving its corresponding weight a fractional value.

The exactness factor is a value between 0.0 and 1.0 and represents a cut off point in attribute similarity calculations. A similarity value is a number between 0.0 and 1.0; 0.0 meaning 'no match' and 1.0 meaning 'identical'. If an attribute-attribute similarity calculation produces a result less than the exactness factor that result is automatically set to 0.0 (no match). If the result is above the exactness threshold, it is allowed to retain its value.

The end-user modifies these values through one of two interfaces. The 'advanced interface' consists of a slider for each attribute factor, allowing accurate manipulation of weighting and exactness. The 'simple interface' is a set of three radio buttons for each attribute that break the possible factor values down into discrete parts, 'low', 'medium' and 'high'. This second interface has been produced in response to user trials, where the modification of factor values using the advanced interface was not intuitive enough for some users; there was too much control over values when it was not immediately obvious what they did.

Interaction with the Q-Space itself takes place using the 3D DIVE interface, which contains a HUD. Individual object status and progress information appears at certain locations within the HUD. It also has the added feature of being able to control the labelling of search items that make up the Q-Space. All of the items within the space can be made to displace the value of a selected schema attribute, such as 'author' or 'title'. The goal of this feature is to be able to see all the values of an attribute at a glance and find the one you are looking for, rather than having to select each item individually and read its details when they are displayed in the HUD.

Movement around the 3D space has also been adjusted to suit abstract landscapes by using the two-mode 'DIVE vehicle'. The first mode is 'point-and-click', which supports the selection of individual items in the space. The second mode is the 'browse' mode, which allows a user to navigate around the abstract space, but remain fixed on a previously selected point. This point acts as a landmark, albeit an 'aggressive' one as it never moves from its fixed point on the screen.

The final feature worth mentioning in the abstract demonstrator is the keyword list. When a user creates a new query, they are presented with a dialog box and a text field to fill in. They also have the option of viewing all of the queries currently cached in the repository and can select one of them from a menu. This allows an end-user to see what other queries have been issued by people and which queries are already in the cache and can therefore be processed quicker – there would be no delay as the Data Extractor parsed the text being returned from OPAC, as a keyword from that list is guaranteed to be in the repository.

Conclusions

In this section we provide a critique of the architecture outlined above, by drawing the conclusions from the last set of user studies [Mariani and Crabtree] that

utilised the demonstrator. We shall also touch on issues and ideas raised in [Mariani et.al.]. Based on the findings from this section, the next section will outline the requirements for a new Q-Space generator architecture. We begin with the recommendations from the user studies and how they impact on the existing architecture and how a new architecture might support them.

Issues raised in user studies

The findings of the user studies and suggestions for solutions are detailed elsewhere within this deliverable [Mariani and Crabtree]; we will only provide a brief review of them here, but we will look at how the architecture of the demonstrator has caused these issues to arise. What follows is a discussion of the issues raised and their causes within the architecture.

The reliability of the system is a big issue, as pointed out in [Mariani and Crabtree]. The demonstrator could crash easily, usually due to the performance of the DIVE visualiser rather than the Q-PIT. The example that is cited is the 'timeout' problem, where DIVE failed to load an object and then tried to manipulate the object, resulting in a system freeze. The Java-DIVE interface (JDI) [Stenius and Trevor] that the Q-PIT component has used in the past to add objects into a DIVE world, degrades with 'sustained and rapid communication'. The current architecture does not shy away from issuing sustained or rapid communication; consider the communication required to visualise the object movement to the Benediktine space or the updates needed to animate the force displacement of a Q-Space. Couple this with the scalability concerns that were raised in the user trials and it is very apparent that, to address these user issues, the visualiser must find an alternative method of communication between Q-PIT and DIVE. The development of Jive, a Java-DIVE native interface again described in [Stenius and Trevor], does attempt to address the performance of Q-PIT/DIVE communication but it too scales badly when handling a large number of event callbacks. In addition it suffers, like JDI, through the lack of support for Tcl scripting, which is crucial if we are to create interactive worlds and graphical user interfaces under the direct control of the visualiser rather than the Q-PIT. Jive does not currently provide access to the complete set of DIVE libraries, only the core networking and distributed database components. The lack of support for DIVE modules also raises implementation issues for interfaces utilising innovative interaction devices, such as PDAs or tilt boards, which tend to be created as DIVE modules.

The Q-PIT used in the demonstrator has been modified to calculate groupings (clouds) based on similarity measures, information that is not returned from the OPAC. It is surprising then that, despite this extra processing effort for an undoubted gain in legibility, for some search items the *full title is not available*, having been truncated by the Q-PIT Java code. The decision to impose an attribute field limit of 50 characters is understandable; storing a field value as an array of characters rather than as a Java String object allows the Q-PIT code to

access the field value faster, which is particularly important for the Benediktine domain and similarity calculations. On further inspection though it is not as efficient in its use of memory. When using the Library demonstrator, the OPAC data source has seven attributes. This means that for each search item there are seven fields, each field being an array of 50 characters. Of those seven attribute values, four of them are usually under 12 characters in length:

ISBN: 10 characters
Classmark: 5 to 7 characters
Published date: 4 (occasionally 9) characters
Publication type: 4 ('book') to 19 ('oversized pamphlet') characters

Memory usage is certainly an issue, especially as the Java-based Q-PIT is having to store both the search item information and the calculated domain and similarity values so it does not have to recalculate the values or re-query the Data Extractor. When scaling up to handle larger sets of results the Q-PIT can rapidly run short of memory space to allocate for these values.

Relating separate searches requires the Q-PIT to be modified to support the co-existence of Q-Spaces. In the year 2 Q-PIT architecture this is not possible because many of the functions and variables that are used in the generation of a Q-Space, in addition to the functions that are used by Jive, are declared as static or global functions. There is no way to separate a Q-Space from the Q-PIT and visualiser that it is being rendered on; the implementation mixes data management, processing and visualisation together in one component of the architecture. To complete the support needed for the *relating separate searches* feature a Q-PIT must be able to take parts of multiple Q-Spaces selected by the end user and combine them into a new Q-Space. Internally this would require a Q-PIT to assign unique identifiers to every search item it contained and be able to tell which items were marked for including in the new Q-Space and which were not. The new Q-Space would only contain the selected items from the original parent Q-Spaces. It would be a non-trivial task to take a Q-Space and try and calculate the keyword that should generate that set of results.

An opposite of searching by 'select and join' is *searching by elimination*. [Mariani and Crabtree] suggest that deletion operations can be performed on objects and groups of objects and when an artefact is deleted it remains so, even if the Q-Space is regenerated. The cheapest way to do this would be to mark deleted search items by setting a flag on that item's representation within the Q-PIT. When the space was next generated the Q-PIT would have to check each item to see if it had been flagged as deleted or not. If the item has been deleted then it should perform no further part in the Q-Space generation process. A more elegant solution would be for the Q-PIT to have better memory management within itself. When an object is deleted from the visualisation of the Q-Space, it should be deleted from the Q-PIT in-memory representation too. There are two advantages to this method; the first is that the memory occupied by the deleted item(s) can be recycled. The second is that the Q-PIT does not have to check a flag on each item

when it has to regenerate the space, it can just get on and do it, safe in the knowledge that all the items in memory are items that are to be processed and sent on to the visualiser for display.

To support *individual items and classmark relations* the Q-PIT and Data Extractor would have to be modified. The existing telnet and web-based versions of the OPAC system offer this search facility. The Data Extractor could be modified to send classmark requests, in addition to the standard keyword requests, and parse the data returned as a result. It is likely that a slightly different parser would have to be used to capture the results from the page-based text that is returned by OPAC, although the same repository could probably be used. The ability to do this also relies on the fact that the Q-PIT can manage at least two Q-Spaces at once; one for the original molecule, the other for its classmark relations.

The position of a Q-Space molecule is calculated by the Q-PIT process, the visualiser has no control over the final position. This is again due to the mixing of data, management and presentation concerns within the Q-PIT, which leads to *overlapping objects*. Ideally the Q-PIT should be able to generate a generic Q-Space and pass that on to the visualiser. It is then the responsibility of the visualiser to work out how best to display the Q-Space because it should have better access to information such as the size of the window containing the 3D world and the end-user's 3D position and orientation. The user would also have more control over where they positioned the Q-Space; it could be placed where they stand, at a fixed distance in front of them or at a position they are somehow indicating towards. This would address another issue from the user trials; the *centred view*. In order to centre the molecule in the user's visualisation window the details of the users position and window size have to be known. As we have pointed out, the visualiser itself is in a better position to access that information than the Q-PIT. Both of these features become increasingly important when you return to the prospect of multiple Q-Spaces in one visualiser. The placement of new Q-Spaces becomes very important and needs to be under the end-user's control if they want to create and maintain a space in which they feel comfortable working. The Q-PIT does not and indeed should not need to know the absolute position of a query molecule in a space, which is by its very nature, abstract. All that the Q-PIT should know about is the fact that the molecules are represented in the space somewhere.

If we are to create and maintain the separation of data processing from data visualisation the visualiser must take on some of the role of managing the interaction between the end-user and the Q-Space. Ideally the visualiser should be able to display the results in any way that it wants to, without affecting the operation of the Q-PIT. Issues such *as clicking on objects within a cloud*, *difficulty of the concept of weights* and the *positioning of object text* in space are presentational issues that demand some work on the representation of the Q-Space within the visualiser. In addition, the issue of *returning to an object* in a Q-Space should be managed by the visualiser rather than Q-PIT, as this need stems from an end-user's interaction with the data, not the data itself. To draw a parallel with the

World Wide Web, to get back to a web page you were previously browsing you press the 'Back' button. The web server that is serving you the data does not know where you have been, and in most cases it does not need to; it is the browser that manages the history information of a user. This information is unique to each user and is not connected to the data that the user is viewing and so should not be mixed with the processing or management of the data on the server (Q-PIT) side.

Supporting Abstract Landscapes in Future Work

In the 'proactive further work' section of [Mariani et. al.2] several potential futures are outlined, all of which involve technologies from the library demonstrator. We have already touched on extending the Q-Space generation system to support multiple Q-Spaces, but another possibility that we have not yet looked at is the idea of extending the range of accessible data sources beyond the OPAC system. Suggestions included access to Z39.50 servers and searches for books on Amazon.com. In the demonstrator outlined above, this would require some substantial changes to accommodate the new data source. A new Data Extractor would be needed to parse the results from each new data source in addition to a new repository for the Extractor to cache its results. The Q-PIT would have to be made aware of the existence of these new data sources, their network addresses and table names in order to be able to request cached data from them. To be able to compare results from different queries the Q-PIT would need to be able to display multiple Q-Spaces, with potentially each one generated from a new data source. The internal architecture of the Q-PIT itself would benefit from being able to configure itself for any data source that a user selects. The Q-PIT code used in the demonstrator has tried to be generic, but in the sense that it is easy to change and *recompile* the code to support a new data source. By far the better solution is a system where a Q-PIT can request the schema of a selected data source and configure itself to handle that schema. This would undoubtedly be at the expense of performance, the compiled code would run faster because, for example, it knows how many attribute fields to expect from the data source that it has been compiled for.

The Cyberpond is also described in [Mariani et.al. 2]. The pond together with the PDA devices, are potentially two alternative interfaces to an abstract Q-Space. Yet again we see another argument for the separation of data processing from visualisation. Trying to port the demonstrator architecture verbatim to a PDA would not be feasible. It would not be possible to run a visualiser program that included a Q-PIT on a PDA and use it successfully. Separating the processing from the display offers obvious benefits in the case of the PDA and also opens a further avenue of exploration. If a Q-Space can be generated by a Q-PIT for a range of alternative interfaces, we could capture some of the interaction between Q-PIT and the visualiser and broadcast it to other users browsing the same data source. This cross platform awareness coupled with a simple messaging system would allow user to user interaction in an abstract space, where a consistent view

of the space across the two users is not essential, indeed for the users who are connecting via different devices a consistent view is not possible.

There is a final aspect that has only been touched on – that of the Data Extractor and mySQL repository. Whilst separating visualisation from data processing, we might also wish to separate data processing from data capture and storage. The demonstrator uses the Data Extractor as a source for a query identifier to be used with the repository. It does not use it as a source for the results itself. This is unfortunate, considering the Extractor's relatively small role in the production of the Q-Space. Once it has loaded the repository and returned the query ID, there is nothing for it to do until the next query keyword arrives. The technology that is used by the Data Extractor to load records into the mySQL repository has to be replicated within the Q-PIT so that is can retrieve the cached search items. Ideally we would like to hide the implementation of the repository from the Q-PIT and use the JDBC access technology purely in the Data Extractor. There are several benefits besides hiding the cache implementation from the Q-PIT component. The management of access to the repository becomes easier. The PC that the year 2 demonstrator just described ran on had to be given permission to access the repository, which might not sound like too much of a problem, but imagine if there are multiple Q-PITs out on the network, each decoupled from its presentation. The task of updating the database security system becomes problematic. By moving all of the JDBC access to the Data Extractor, only the machine that the Extractor is running on needs to be authenticated.

Another benefit of removing the JDBC code from the Q-PIT is that the physical memory consumed by the Q-PIT is lessened. In the already process intensive sequence of tasks required to generate a Q-space, the use of JDBC access to retrieve query records added an extra overhead both in terms of time and memory requirement. The loading and registering of the JDBC driver added an extra, albeit slight, time delay in the Q-space generation process, but the bigger issue is with the in-memory storage of the search items. When an SQL command is executed via JDBC the results from the query are loaded into memory as a JDBC ResultSet object. The Q-PIT can only access individual items in the ResultSet object at one time. The Q-PIT reads each search item from the JDBC ResultSet object and creates own copy of the search item (i.e. in the demonstrator it will create seven character arrays of length 50 and place each field of the search time into the corresponding array) This duplication leads to massive memory consumption for larger result sets, as there are at least two copies of the result set in memory at some point. It is far more efficient to read the search items from the repository a row at a time, allowing the results to be added one by one, but this would mean sending a new SQL command to the repository to fetch each line. By returning the search results from the Data Extractor around the socket interface to the Q-PIT, the Q-PIT can read results from the socket one at a time.

Conclusions

The main conclusion that we can draw from the brief discussion above is that the segmentation of the architecture into several specialised components is desirable so as to be able to realise the other issues more simply. We can identify three main segments that should be addressed in the new architecture:

Data Capture: Separating the Data Extractor and its cache from the Q-PIT implementation. This allows greater flexibility in the management of the database that is used as a cache. The separation from the Q-PIT also means that, if the data source changes its representation of the search results, the Data Extractor can be adapted to cater for that change, rather than having to modify the Extractor and the Q-PIT code, as would currently be the case. This is also true when it comes to adding new data sources. It may be the case that a different data source would suit a different implementation of a repository. The separation that we should create would allow that to happen with no impact on the Q-PIT code.

Data Processing: Separating the Q-PIT from the Data Extractor and Visualiser is fundamental to the future development of the abstract landscapes like those outlined in [Mariani et. al.2]. The Q-PIT must not interact with the user directly but use the visualiser as its agent. Internally the Q-PIT should be modified to remove information which is purely of a presentational nature. It is too easy to mix the processing of data with its presentation, as seen in standards such as HTML. If alternative interfaces are to be supported the Q-Space that the Q-PIT generates must be generic enough to allow that interface to visualise the information in its own way. By embedding somehow, presentational information within the Q-Space data it constrains the visualiser's rendering of that data.

Visualisation: Any new visualisation software must be able to take a generic Q-Space and display it. The visualiser must be able to communicate both with the Q-PIT that it requests the data from and the end-user. The Q-PIT must not be able to interact with the user directly, so the visualiser must be able to collect the information that the Q-PIT needs in order to be able to operate correctly. Although is means extra work for the visualiser, it does mean that a designer is free to explore different representations of the Q-Space and the interactions with the user without having to be weary of making changes that will affect the Q-PIT itself.

Once the architecture has been partitioned into a structure similar to that just listed above, it becomes possible to develop components of the structure independently and make the modifications necessary to address some of the issues raised in the user trials. One of the more important requirements appears to be the facility for generating more than one Q-Space at once. The segmentation of the architecture should make this goal more achievable, either though a new Q-PIT or a visualiser that can connect to more than one Q-PIT at once. The ability to easily include new data sources within a Q-PIT is also a desirable feature. This requires that the Q-PIT be as generic as possible, something which it cannot do when it is tightly coupled to the visualisation of a Q-Space.

Requirements for a New Q-PIT Architecture

Based on the features and drawbacks that have been described above, here is a list of requirements for a new Q-PIT architecture.

- Separate presentation and data processing. Detach the Q-PIT from the visualiser and use a new method of communicating between the two rather than Jive/JDI. This allows new visualisers to be created, and means that, in the future, changes to one component will not impact on the other component.
- Separate data processing from data capture and storage. Detaching the Q-PIT from the query results repository affords easier management of the repository itself and makes the addition of new data sources easier. The Data Extractor should be the sole client of the repository.
- Support for multiple Q-Spaces within one Q-PIT. This appears to be crucial to the success of the classmark relation issue as well as for visualisers such as the CyberPond.
- Support for classmark as well as keyword queries when using the OPAC data source. This assumes the requirement listed directly above.
- The Q-PIT should be as generic as possible so as to be able to support new search sources (via a new Data Extractor) with different schemas. Included in this genericity is the ability to run on more than one platform.
- Browsing by elimination must be supported, as highlighted from the previous set of user trials.
- The Q-PIT must be scaleable and robust.

The New Q-PIT Architecture

Before outlining the new architecture, we shall briefly look at the new procedure of generating a Q-space. The creation of a Q-space can be divided into three broad processes.

- *Data collection*: gathering data in response to a user's keyword
- *Processing*: forming the Q-space from the data that matches the keyword
- *Presentation*: rendering of the resultant Q-space

Each of these steps can require, in certain scenarios, intensive processing to produce the results that are required by the following step. The production of a Q-space from data collection through to visualisation is essentially a serial process; there is little opportunity for parallelism across or within each step. In previously

reported prototypes of the Library demonstrator, nearly all of these steps were performed on one machine. In the new architecture there is scope for multiple machines to be involved in the overall generation of a Q-Space. This might appear to be a contradiction based on the statement that we have made that there is very little parallelism in the overall system. By moving the Q-PIT executable to a different location we free up CPU time on the visualisation machine allowing it more time to attend to its interface and user interactions. Listed below are architectural components of the new Q-PIT architecture.

Data Source

A data source is the actual third-party resource that supplies the results in response to an end-user keyword. Examples of data sources would be the OPAC system or Amazon.com book store. A typical data source is a web page with a search facility on it that runs a CGI script on web server when invoked with the correct parameters. A structured result page that is returned in response to a query is also useful (it makes parsing the page a lot simpler). A data source must have several external properties to make a good Q-Space, for example, results returned from Amazon.com search can be grouped by price, publisher or author. Data sources that do not have many external properties are not good candidates.

Data Extractor

The Data Extractor acts a bridge between a specific data source and a Q-PIT component (see below). The Data Extractor has the role of gathering together search items in response to an end-user query. When it receives a search request from an end-user it checks a ‘cache’ of previously searched keywords to ascertain if the results are already contained in the repository. If the results are found in the repository the Data Extractor returns these results as a response to the query request. If the results are not found then the Data Extractor formats a query based on the URL of the data source (it mimics a URL that invokes the CGI search script on the web server of the data source). The results that are returned by the data source are parsed by the Data Extractor and are loaded into the ‘cache’. The keyword is also saved to indicate the presence of its associated results in the repository. The Data Extractor then returns the new set of results as a response to the query request. Once this process has completed the Data Extractor takes no further part in the (re)generation and management of a Q-Space.

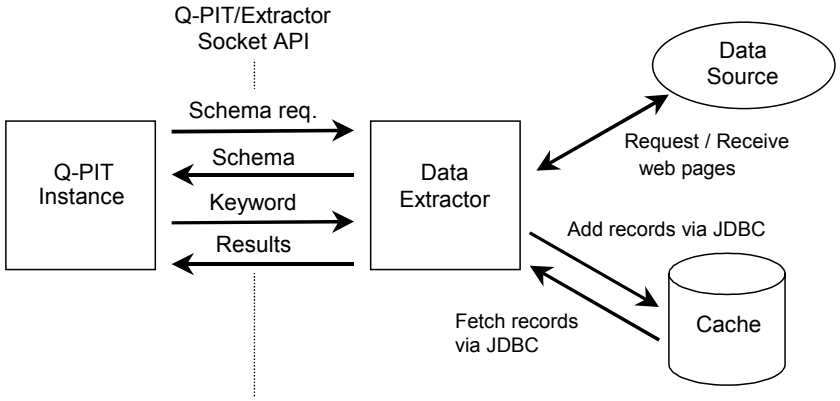


Figure 3: The separation of data processing and data capture

Figure 3 shows the Q-PIT / Data Extractor side of the new architecture. The Q-PIT no longer has direct access to the cache, simplifying its implementation. All of the JDBC code is now contained in the Data Extractor, where is is used for loading and reading records. The sequence of events as shown in figure 3, makes it clear that the implementation of the cache is completely separate from the implementation of the Q-PIT, in fact if it was not for the delay in parsing the web pages in the first place, the Q-PIT cannot tell if the results are already in the cache, being parsed for the first time or if a cache even exists.

In addition, database management is simplified, with only one database client (the Data Extractor) needing to be given security clearance to access the cache database. The communication between the Q-space generator and the Data Extractor takes place via a socket interface and simple API, allowing the Q-PIT to be completely remote from the Data Extractor and its cache. This separation and the introduction of the simple API has allowed the architecture for the Q-PIT to become more generic, supporting any Data Extractor or data source that conforms to the Q-PIT/Extractor API. This is achieved by having a message in the API that a Q-PIT instance can use to request the schema of the data source from a Data Extractor. A Q-PIT instance can configure itself for this schema before receiving any search results from the Extractor. Figure 4 shows a sample of a typical dialog between a Q-PIT and the OPAC Data Extractor.

```
    Sender      Message                    # Comment
     QPIT:      get_meta_data              # metadata command
Extractor:      7                          # number of attributes
Extractor:      classmark VARCHAR          # 1st attribute name and type
Extractor:      pub LONGVARCHAR            # 2nd . . .
Extractor:      pubdate LONGVARCHAR        # 3rd . . .
Extractor:      pubtype LONGVARCHAR        # 4th . . .
Extractor:      author LONGVARCHAR         # 5th . . .
Extractor:      title LONGVARCHAR          # 6th . . .
Extractor:      isbn LONGVARCHAR           # 7th . . .
```

Figure 4: Metadata discovery

In response to a `get_meta_data` request from the Q-PIT, the Data Extractor first returns the number of attributes (fields) in the schema and then the attributes themselves, one per line. By returning the number of attributes first, the Q-PIT can know how many of the following lines to read in. A similar technique is used to retrieve the results themselves. The Q-PIT sends a keyword search message to the Data Extractor, which loads the results into the cache if they are not already there. The Data Extractor then writes the number of records to be expected to the socket, which the Q-PIT reads and uses in reading the results. The Extractor writes the specified number of results directly to the socket; the order of the attributes of a record matches the order of the attributes returned in response to a *get_meta_data* query. The Q-PIT reads these results from the socket, a result at a time and places them in memory before closing the socket interface to the Extractor. This new architecture has the distinct advantage of letting Q-PIT build up a list of results in memory by streaming a record at a time into an internal buffer. Recall that previously the JDBC version had forced the Q-PIT to hold all the search items set in its local memory before being able to access a single item.

The computational overheads of all the steps involved in the generation of a Q-space has already been noted and to this end the Data Extractor, via the Q-PIT/Extractor API, issues status messages to the Q-PIT. These messages, which the Q-PIT passes onto the user's client, indicate how far through the query retrieval process the Extractor is. This is especially useful in situations where a query has been issued that is not in the cache and a large number of web pages have to be parsed to be able to collect the entire result set.

MySQL Database and JDBC

The single 'cache' of keywords and associated results is implemented by using a mySQL [mySQL] relational database. A schema is associated with each data source that a Data Extractor can talk to. It is possible for two Data Extractors to share the same database management system; they have different tables within the system. This removes the need to have one complete installation of mySQL per Data Extractor.

A JDBC [JDBC] interface library is employed to the Data Extractor to access the database. No other architectural components have access to the mySQL database.

Q-PIT Server

This component spends most of its time listening to a socket awaiting connections from end-user clients. When a connection is received the server creates a new instance of a Q-PIT (see below) and passes it to the socket. It then goes back to listening for more connections.

Q-PIT

A new instance of a Q-PIT is created when an end-user's client connects the Q-PIT Server. The Q-PIT communicates with the client across a socket using a simple text based API. This is the only method of communication with the Q-PIT, it does not present a graphical interface to the user directly.

Before a user can issue a search request they must identify themselves, by providing the Q-PIT instance with username (this can be machine generated if the Q-PIT is talking directly to an automated visualiser such as the CyberPond, where the end-user does not actually invoke the search request). The type of client that is being used to talk to the Q-PIT must also be identified. This allowed the Q-PIT instance to tailor its processing and message syntax to support that particular client. Once the client has provided a name and a client type, it can request a list of data sources that the Q-PIT instance can communicate with. The Q-PIT sends a list of these data sources in response to the 'list_sources' message from the client. The client responds with the name of the data source that they with to issue a search on. The Q-PIT acknowledges the selection and awaits the end-user's keyword search. Once the user issues a search request the Q-PIT opens a socket interface to the Data Extractor that corresponds to the user's selected data source.

The processing of search results that are returned in response to an end-user's keyword takes place in a Q-PIT, modified to support the calculation of data clouds (the grouping or region features of a Q-Space). The algorithms used to generate the Q-Space from these results are explained in detail in [Mariani and Colebourne], but a brief summary of steps required has been included below.

1. Generate a Benediktine space [Benedikt] for initial "random" positioning of points within the space
2. Generate a similarity matrix
3. Generate the minimal spanning tree
4. Identify regions within the tree
5. Apply a force displacement algorithm

Note that the actual production of a hull to represent the data cloud is no longer part of the processing of a Q-PIT, its production has become the responsibility of the visualiser. After the completion of all of the processing steps the final Q-Space is sent across the socket interface to the client using the Q-Space API. Throughout the production of the Q-Space, from the reception of the initial client keyword, via the dialog with the Data Extractor to the final displacement algorithm, status messages are returned to the client to inform the end-user of the progress of their query.

The client, on behalf of the end user, may choose to regenerate the space at some later point through the use of additional messages from the Q-Space API. These messages and a more complete description of the Q-Space generation process are listed below. The socket interface is never closed until the client drops the connection, which in turn causes the Q-PIT instance to gracefully close down and unload itself from memory.

Figure 5 shows how all of the components of the new architecture fit together.

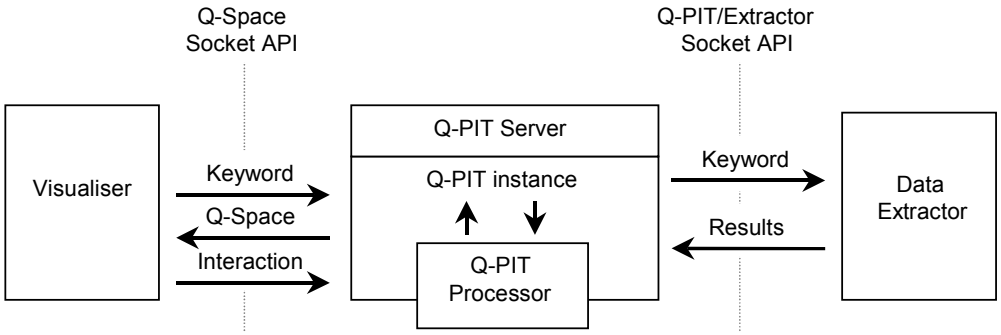


Figure 5: The new architecture.

The New Q-Space generation process

Connecting to the Q-PIT server: The Q-Space API

Before walking though the sequence of steps required to generate a new Q-Space, it is worth describing the method of communication between the end-user's visualiser and the new Q-PIT. The Q-Space API is the interface between the data processing and visualisation. Some of the operations that were performed by the Q-PIT in the original architecture have become the responsibility of the visualising client, so are not described here. The information that the client has to work with comes from the Q-PIT and consequently has to pass across the Q-Space API interface. The API is designed to be human readable and straightforward to parse.

The API offers a mechanism for a client to request and manage a Q-space that has been created on their behalf and resides in Q-PIT's memory. These operations are issued by the client in response to specific user activity and help to synchronise the Q-space copies in client and server memory. It is worth noting that, for interfaces running on low memory devices, Q-PIT acts as an extended memory for that device. Included below is a list of the primary functions of the API:

Initial Handshake: When the client and Q-PIT first come into contact with each other, the client sends a 'hello' message indicating its type, such as

'Dive'. If the Q-PIT recognises the client type it modifies its processing and messaging syntax to suit that client.

List Data Sources: The client issues this command and receives a list of data sources that are currently available within the Q-PIT. Each data source in the list actually corresponds to a Data Extractor that the Q-PIT will communicate with, although the client is unaware of this. Once the client has the list of data sources it will display them in some fashion to the user, who will select one. The selected data source name is returned to the Q-PIT, which can begin to set up a socket interface to the corresponding Data Extractor.

Schema: There are two operations that can be performed with a schema message. The first is to inform the client of the schema and mapping of the data source that they have previously selected, allowing the client to tailor its user interface to support the list of attributes that are returned. This is crucial for the client when setting up the mapping or weighting interface, where the user can modify variables that are used to calculate the Q-Space. Once a user has changed these variable settings they are sent back to the Q-PIT. This is the second use of the schema message and allows the Q-PIT to set the mapping and similarity variables in memory.

Query: A query message is issued by the client when the end-user wants to (re)generate a Q-Space. To create a new Q-Space the query message contains a keyword value, which is the keyword used in the search. A query message can also contain mapping and weighting information, similar to that found in the schema message. The difference being that, when these values are included in a query message, it forces the recalculation of the Q-Space. The inclusion of this information in a schema message does not force the recalculation of the space but merely sets the variables to be used next time the space has to be recalculated. It is also possible to delete queries from the Q-PIT's memory.

Update or Remove an Object, Edge or Group: Each item that constitutes a Q-Space, be they record objects, links between records or region (grouping) data is assigned a unique identifier within the Q-Space. The client and Q-PIT use the identifier to manage various features of that particular item, such as its position, colour or membership of a group. It is particularly useful for updating an object in a Q-Space, such as a record because the updated information is all that needs to be sent. The client already has a copy of a record so it would be inefficient to send an object's complete details every time some part of that detail changed. The client can use the identifier values to inform the server that an item has been removed. When a user selects and removes an item, be it an object or a group of objects, a message is sent back to the Q-PIT to remove the corresponding objects from its internal copy of the Q-Space. This feature supports the browsing by elimination, highlighted in the previous set of user trials.

In order to begin the process of generating a Q-Space the client has to connect to the Q-PIT. This is done through the Q-PIT server, which is responsible for

generating an instance of the Q-PIT for each client that connects to it. The connection between the visualising client and Q-PIT is done across a socket interface, allowing the Q-PIT and visualiser to run on completely different hosts, easing the processing requirements on the client. Once the client has connected to the Q-PIT server, the server creates a new instance of a Q-PIT and passes the newly opened socket to it. The server takes no part in the dialog between a client and Q-PIT at the level of the Q-Space API. Once the Q-PIT instance has the socket it awaits an initial 'handshake' message from the client

Handshake: supplying the user name and client type

The client can send a user name and client type to the Q-PIT, allowing it to tailor the way it interacts with the client and generates the Q-Space. This self-configuration allows different types of visualiser to use the Q-PIT service effectively. The Q-PIT knows how much of the Q-Space it has to generate based on prior knowledge of a client type's requirements. In the current incarnation of the Q-PIT, there are a number of heterogeneous clients that use the service, each with different requirements. Some clients require the whole Q-Space to be generated, such as DIVE, including the force displacement calculations. Text based visualisers cannot display 3D information about a Q-Space, so consequently the force-displacement calculation is redundant to it. The Q-PIT knows this and can omit the force displacement calculation from its list of functions to perform. Other clients handle the generation of the Q-Space within the visualiser itself. In this case the Q-PIT is nothing more than a bridge between the client and the Data Extractor, but still separates the visualisation from the data capture process.

A typical dialog between the client and the Q-PIT instance would look something like that described in figure 6. In this case the Q-PIT can tell that the client is a Dive client and the user is called 'Peter'.

```
Client:    hello user=Peter type=Dive
Q-PIT:     hello
```

Figure 6: Handshake message between client and Q-PIT

It is worth noting here that the user name is not crucial for the successful operation of the Q-PIT. The user name of every client connected to the Q-PIT server is returned when a client issues a `list_users` message. How the client displays and uses this information is up to the designers of the client. User names do not have to make sense to another user. We recognise the fact that some client implementations may not be driven by the end-user, rather the end-user reacts to information that the client displays. This is the case in the CyberPond, where the pond itself issues queries to the Q-PIT that it thinks end-users will find relevant.

If the client type that a client supplies is not recognised, the Q-PIT reverts to a 'default' message syntax, in effect the same syntax that is used for the DIVE client type.

Listing Data Sources

Once the client has exchanged handshaking messages with the Q-PIT it sends a `list_sources` message. In response the Q-PIT returns a list of all the data sources that it currently knows of and can connect to. Figure 7 shows a sample list.

```
Client:   List_sources
Q-PIT:    List_sources number=2
Q-PIT:    Name=OPAC description=Library+OPAC+system
Q-PIT:    Name=Amazon.com description=Amazon+online+bookstore
```

Figure 7: Listing the data sources

The parsing of this multi-line response from the Q-PIT is aided by the inclusion of the 'number' attribute in the first line returned. The client knows how many lines to expect, with one data source being described per line. How the client chooses to display this data source information to the user, and collect the user's selection is entirely up to the implementers of the client. This is another example of the visualiser/data processing separation that has driven the design of this new architecture. Recall that before, in the architecture described at the beginning of this document, the gathering of user information was done through Java GUI components, under direct control of the Q-PIT and DIVE components build into the visualiser HUD.

Another feature of the API, which is demonstrated in figure 7, is the use of name/value pairs in a message. In general the API tries to begin a message with a message identifier, such as `hello` or `list_sources`, and then send other data as name value pairs; `number=2` or `name=OPAC` for example. The white space between these pairs act as a delimiter character; the equals character acts as a delimiter between the name and the value, name being to the left of the equals character, value on the right. When an attribute (name/value pair) contains a delimiting character within the value, the character must be encoded so as not to confuse the client parser. The method chosen to realise this feature was URL encoding [Berners-Lee], which is used to encode data from a web page form before it is sent to a CGI script on a web server. As a consequence, many implementations of this encoding and decoding process exist, for a wide variety of programming languages and platforms. This made it the natural choice for our API. An example of this encoding can be seen in figure 7. The value associated with the description attribute has had the white spaces converted to '+' symbols.

Selecting the Data Source

When the user has selected a data source that they want to search, the client sends this selection to the Q-PIT as a *select_source* message (Figure 8).

```
Client:     select_source source=OPAC
Q-PIT:      select_source status=ok
```

Figure 8: Identifying the selected source to the Q-PIT

The Q-PIT responds with a single line message confirming that all is well. Internally the Q-PIT looks up the address of the Data Extractor that manages access to the selected source, so as to be ready to communicate with it when the client sends further messages relating to the search.

The Keyword Search

At this point the client can send a keyword query to the Q-PIT, which will in turn pass the keyword on to the Data Extractor for the selected source. Figure 9 shows a typical query from an end-user client.

```
Client:     query keyword=java query_id=555
```

Figure 9: A query command from the client.

Upon receiving this query the Q-PIT opens a socket interface to the Data Extractor, sends the keyword across the socket and waits for a response. The client has an option of specifying the query identifier string (`query_id`) itself or rely on the Q-PIT to generate on for it. In the case shown in figure 9, the client has selected an identifier that it wants to use to refer to the invoked query.

The client's query message invokes a large amount of processing within the Q-PIT to generate the final Q-Space, so the messages returned to the client are a mixture of data and information messages, which contain status information regarding the Q-PIT's current actions. The Q-PIT's first status message is given in figure 10, and is the first of many information messages to be sent to the client.

```
Q-PIT:      info query_id=555 message=Querying+the+data+source
```

Figure 10: A query command from the client.

Note the use of the *query_id* attribute, which informs the client as to which query this message refers to. This is necessary for situations where the visualising client is running more than one search at once.

Collection of data

The majority of the details regarding the structure and communication between the Data Extractor and Q-PIT have been shown already, but we will give a brief description of the process again here. The communication between the Q-PIT and the Data Extractor takes place across a socket using the Q-PIT/Extractor API. This is a very simple API, but allows the Q-PIT to gather all the information it needs, both in terms of the structure of the data source itself and the search items that are returned from the search. The Q-PIT collects the search results in two steps. The first step is to ascertain the structure of the data source, known as its schema. This is the number of attributes that make up an individual search item. In the case of

the OPAC, there are seven attributes; ISBN, author, title, etc. The Q-PIT uses this information to set up its internal data structures that it will place the search results into. Figure 4, in the previous section, shows the dialog between the Data Extractor and Q-PIT for discovering the schema of a data source.

The next step that the Q-PIT takes, after establishing the format of the data to be returned, is to issue the search keyword. As in the previous architecture, the Data Extractor takes this keyword and compares it to all the keywords in the local database cache. If the keyword and its results are not in the cache, the Data Extractor issues the query directly to the data source itself and parses the returned results directly into the database cache. Where the new architecture differs from the old is that the Data Extractor returns the results directly to the Q-PIT, rather than the Q-PIT having to access the cache. The benefits of this approach have already been mentioned above. Throughout this process the Data Extractor is sending back to the Q-PIT status information about its current processing. This is especially useful when there has been a 'cache miss', i.e. the keyword has not been found in the local cache and a direct query has to be issued to the data source. The Q-PIT does not use this information itself, rather it passes it on as an information message to the client that invoked the request.

Q-PIT processing

Once the Q-PIT has read all of the search items that have been returned from the Data Extractor, it closes the socket to the Extractor and begins the generation of the Q-Space. We shall not go into too much detail here of the actual processing steps involved as they are identical to the original Q-PIT processing, save some implementation differences which are covered later. The only noticeable differences are that the 3D geometry for a data cloud is no longer calculated by the Q-PIT itself and that different types of client need all or just part of the Q-PIT process steps to be performed. We shall briefly step through the processing stages, focusing on the message passed from Q-PIT to client, rather than the algorithms used in the steps themselves. The processing steps that the Q-PIT goes through to generate the Q-Space are as follows:

1. Generate a Benediktine space
2. Generate a similarity matrix
3. Generate the minimal spanning tree
4. Identify regions within the tree
5. Apply a force displacement algorithm

Sending the initial Q-space

To get to a stage where it can start to send Q-Space information to the client, the Q-PIT must first calculate the Benediktine space that represents the search items. This involves calculating the domain for each of the attributes in the schema (in the case of OPAC, there are seven attributes). The Q-PIT represents this processing by sending what we have called 'progress bar information' messages

to the client. These messages are particularly suited for showing the status of long and repetitive calculations, and although a progress bar is a good visualisation of this information, the client need not use it in order to get across the meaning of the messages. Figure 11 shows example progress bar messages sent from the Q-PIT instance.

```
Q-PIT:    info query_id=555 progress title=Benediktine+calculations min=1 max=7
Q-PIT:    info query_id=555 progress value=2
```

Figure 11: A query command from the client.

The first message is used to initialise the progress display on the client. It informs the client of the title of the progress message and the number of updates to expect. The second message in figure 11 shows an update to the progress visualisation. Once the update value has reached the maximum value specified in the progress set-up message, the progress visualisation can be removed. Again, note the use of the *query_id* attribute to identify the query that these messages relate to. Progress messages are also sent as the Q-PIT instance calculates the similarity matrix and minimum spanning tree for the Q-Space. As we have already mentioned, these steps are identical to the ones outlined previously and in [Mariani and Colebourne].

Depending on the client type identified in the initial handshaking message, the Q-PIT can return the basic Q-Space as early as the completion of the Benediktine calculations. The Q-PIT represents a Q-Space by the use of 'objects' and 'groups'. An object is an actual search item, and has data associated with it. A group defines something that links together two or more objects. These are either an 'edge' (a link between two objects) or a 'cluster' (a collection of objects that have been found to be related in some way). The Q-PIT supports creation and deletion of 'object' and 'group' items and in addition object items can be modified or updated. Every object or group in the Q-PIT is assigned a unique identifier value that both the client and Q-PIT use in referencing the object when using the Q-Space API.

At this stage of the Q-PIT processing, the Q-PIT itself can begin to send 'create object' messages to the client together with 'create group' messages that will represent the edges that have been found from the minimum spanning tree calculation step. This is shown in figure 12.

```
Q-PIT:    Newobject query_id=555 id=10 x=0 targetx=0.000000 y=0 targety=0.000000 z=0
          targetz=0.000000 time=4 colour=1.0:0.0:0.0/1.0:0.0:0.0/0.0:0.0:0.0/1.0:1.0:1.0/0.1/0.0
          shape=cube
          data=isbn=0X60355166&title=Mastering+Java+1.2.&author=ZUKOWSKI%2C+John&
          pub=SYBEX&pubdate=1998&pubtype=CDROM&classmark=%5B43%2F0316%5D
Q-PIT:    NewGroup query_id=555 id=126 contains=2
          group query_id=555 id=126 object_id=10
          group query_id=555 id=126 object_id=11
```

Figure 12: Q-PIT messages that begin to create the Q-Spaces.

The first message in figure 12 will cause the client to create a new search item artefact in its visualisation. The formatting in this document does not permit us to show the message as it really is; a single line message terminated by a new line character. The second message in figure 12 is a three line message. The first line indicates that the client should create a new 'group' instance and that it will contain two items (therefore it will be an edge between two search item objects).

Updating Q-space

As the production of the Q-Space continues to take place on the Q-PIT and more status messages are produced, it becomes necessary for the Q-PIT to periodically update the 'objects' that it has asked the client to create. In our example walkthrough, the final updates are sent after the completion of the region identification and force displacement steps. Region identification causes more `newgroup` messages to be sent to the client, this time containing three or more members. The syntax is exactly the same as in the second message in figure 12; only the identifier values and the specified number of items in the group are different. Once all the calculations have been completed, the Q-PIT updates the position variables of each item in the Q-Space. Figure 13 below shows two messages. One is the `newobject` message from figure 12, the other is an update to that initial `newobject` message. It demonstrates that, through the use of object identifiers, there is no need to retransmit a complete object (search item) description to the client every time it changes.

```
Q-PIT:    newobject query_id=555 id=10 x=0 targetx=0.000000 y=0 targety=0.000000 z=0
          targetz=0.000000 time=4 colour=1.0:0.0:0.0/1.0:0.0:0.0/0.0:0.0:0.0/1.0:1.0:1.0/0.1/0.0
          shape=cube
          data=isbn=0X60355166&title=Mastering+Java+1.2.&author=ZUKOWSKI%2C+John&
          pub=SYBEX&pubdate=1998&pubtype=CDROM&classmark=%5B43%2F0316%5D
Q-PIT:    updateobject query_id=555 id=0 x=0 targetx=0.067695 y=0 targety=-0.126619 z=0
          targetz=0.030589 shape=cube time=7
          colour=0.56078434:1.0:0.0/0.56078434:1.0:0.0/0.0:0.0:0.0/1.0:1.0:1.0/0.1/0.0
```

Figure 13: Comparing create and update messages.

There is no need for an 'update group' message as the Q-PIT no longer calculates the positions of grouping artefacts – which is now the responsibility of the client. Groups change when the space is recalculated. In that instance the grouping objects are deleted by the Q-PIT using 'delete group' messages.

In keeping with the idea that visualisation should be separate from data processing, the Q-PIT can make no assumptions about the 'size' of the Q-Space that it generates. It is not appropriate therefore to include absolute co-ordinates in the x, y and z attributes of the Q-Space. In order to best represent the Q-Space within the Q-PIT and allow the visualiser to display it as quickly and efficiently as possible, the co-ordinates that are sent back to the client are bounded within a certain space. This space is a cube, whose centre is at the origin and whose extremities are at 1 and –1, in other words a cube whose sides are of length 2. By centring the Q-Space within this cube we provide an easy way for the client to

position this Q-Space. Recall that in the user trials, issues were raised about Q-Space size, positioning and overlap. By packaging a Q-Space up in this way the client can apply scaling and offset calculations to centre the Q-Space in front of the end-user and at a size where they can see the whole Q-Space molecule. The client is much better placed to do this sort of work than the Q-PIT, which has no information about user position in the abstract space or the user's field of view. It also allows the client to offer the end-user the opportunity to scale and place Q-Space molecules in an interactive fashion, facilitating better user management of the overall abstract space.

Creating another query across the same socket

Having finished one query, the client may want to change the way that query is mapped or produce a new one to go along side it. We shall consider how the Q-PIT and API handle re-mapping shortly, however we shall now briefly explore the way in which additional Q-Spaces can be added.

Rather than force the client to open a new socket to the server in order to generate additional Q-Spaces, a client can create new Q-Spaces by using the currently open socket. The way that this is handled is very simple; the client only has to send a new `query` message, with a different `query_id` attribute value than before (or let the Q-PIT generate a new one). This causes the Q-PIT to create a new internal Q-PIT processor and map it to the new query identifier, so that incoming messages are routed to it, rather than the original query. This feature is very scalable and appears only to be limited by the memory that can be allocated to the Q-PIT. If the client sends a query message with an identifier that is identical to the original query, the Q-PIT will interpret this as an operation on the existing query and force a new search, via the Data Extractor. This feature is useful if the data source that is being searched changes frequently.

Changing the space parameters

In the new architecture the client is responsible for displaying and collecting the parameter data, which is used to (re)generate the space from the end-user. The client can retrieve the current parameter information through the use of the flexible *schema* message. The client can invoke the `schema` message, shown in figure 14, at any time on any data source, whether they have registered for it or not. This feature allows the client to offer the end-user a chance to browse the schema information of any data source that is listed by the Q-PIT in response to a `list_sources` message.

```
Client:   schema source=555
Q-PIT:    schema source=555 number_of_attributes=7 x=pubtype y=pub z=classmark
          shape=author colour=title
Q-PIT:    source=555 number=0 attribute=pubtype type=text factor=1.0 exactness=0.0
Q-PIT:    source=555 number=1 attribute=pub type=text factor=1.0 exactness=0.0
Q-PIT:    source=555 number=2 attribute=classmark type=text factor=1.0 exactness=0.0
Q-PIT:    source=555 number=3 attribute=author type=text factor=1.0 exactness=0.0
Q-PIT:    source=555 number=4 attribute=title type=text factor=1.0 exactness=0.0
Q-PIT:    source=555 number=5 attribute=pubdate type=text factor=1.0 exactness=0.0
Q-PIT:    source=555 number=6 attribute=isbn type=text factor=1.0 exactness=0.0
```

Figure 14: Schema request and response messages.

The Q-PIT, upon reception of the first message in figure 14 from the client, sends a `get_meta_data` request to the Data Extractor for the source that has been specified or currently selected. The dialog between the Q-PIT and Data Extractor has already been shown in figure 4. The Q-PIT reformats this into a message that the client can understand and adds the mapping information that it is currently using to generate the space. This new message is returned to the client; the second message displayed in figure 14. Designers of clients can use this feature in the dynamic creation of an interface to display the mapping information that is being used for a Q-Space. It is also useful for formatting the display of information about a particular search item that a user may select. The schema message is, in some cases, sent by the client to get the current parameter and mapping values of the Q-Space generator that it is connected to. The client can also use the schema message to set parameter values, as we shall see below.

The default mappings to intrinsic and extrinsic Benediktine space attributes are chosen by the Q-PIT in the reverse order that they are returned from the Data Extractor. Figure 14, shows that the x values are created from the publication type (pubtype), y values from the name of the publisher (pub) and z values from the classmark. There is a good reason for doing the default mapping in this 'reverse' order. The first attribute to be returned from the Data Extractor is the ISBN number that, by the nature of ISBNs, is guaranteed to be unique. In fact the mySQL database cache uses this attribute as the primary key. If we were using this attribute in a classic Benediktine space or a cluster based Q-Space, this attribute would not yield any high similarity values because each search item's ISBN would be unique. Using the values from the bottom of the Data Extractor list, which is read straight from the JDBC results, there is more likely to be a similarity between them. In figure 14 above, this is definitely the case.

To change the parameter and mapping values used in the generation of a Q-Space, a client must first display to the end-user what these values actually are. The client does this by sending a schema message to the Q-PIT and parsing the results into a display that allows the users to modify the values in some way. Once the user has modified some or all of these values the client sends the changes back to the Q-PIT. The client has two methods of achieving this. The first method uses the `schema` message to return new values. Figure 15 shows this in action.

```
Client:   Schema source=555 x=isbn y=isbn z=isbn classmark=1.0,0.5
Q-PIT:    Schema source=555 number_of_attributes=7 x=isbn y=isbn z=isbn shape=author
          colour=title
Q-PIT:    Source=555 number=0 attribute=pubtype type=text factor=1.0 exactness=0.0
Q-PIT:    Source=555 number=1 attribute=pub type=text factor=1.0 exactness=0.0
Q-PIT:    Source=555 number=2 attribute=classmark type=text factor=1.0 exactness=0.5
Q-PIT:    Source=555 number=3 attribute=author type=text factor=1.0 exactness=0.0
Q-PIT:    Source=555 number=4 attribute=title type=text factor=1.0 exactness=0.0
Q-PIT:    Source=555 number=5 attribute=pubdate type=text factor=1.0 exactness=0.0
Q-PIT:    Source=555 number=6 attribute=isbn type=text factor=1.0 exactness=0.0
```

Figure 15: Schema modification and confirmation messages.

In figure 15, the x, y and z attributes are all created from ISBN numbers and the exactness parameter for the classmark has been raised from 0.0 to 0.5. The Q-PIT responds with a the standard `schema` message reply, but updated for the changes sent by the client. The Q-PIT will use these new parameter values when it next has to recalculate the Q-Space.

```
Client:   query query_id=555 x=isbn y=isbn z=isbn classmark=1.0,0.5
```

Figure 16: Forcing a recalculation of the Q-Space.

To force a recalculation of the Q-Space as soon as new parameter values are received by the Q-PIT, the client sends the new values in a query message, shown in figure 16; this is the second method available to the client. Regenerating the space causes the Q-PIT to reprocess the similarity matrix, minimum spanning tree, region identification and force displacement (if the client type identifies that it needs to). As this is not a new query, merely a 'reshaping' of an existing one, the Data Extractor does not need to be contacted and the Benediktine calculations do not need to take place. Consequently the search items being visualised on the client can be reused, which means sending *updateobject* messages to the client at the appropriate time. Grouping artefacts in a regenerating Q-Space are no longer valid so must be deleted from the client. The Q-PIT achieves this by sending a `remove` message to the client.

Removing object, groups and Q-Spaces

Figure 17 shows two kinds of remove message. The first removes the item identified by its object identifier. This item could be an object, an edge or a grouping object (usually visualised as a cloud). The second message in figure 17 is a special way of removing all the objects that constitute a Q-Space. This is particularly useful when a new keyword has been issued but the query identifier has been retained.

```
Client or Q-PIT:    remove query_id=555 id=10
Client or Q-PIT:    remove query_id=555 id=_all
```

Figure 17: Removing named or all objects.

It is worth pointing out that either the client or the Q-PIT can send these messages. When they are issued by the Q-PIT it is usually because the Q-Space is being regenerated. By allowing the client to send these messages we have a mechanism for supporting query by elimination, an issue that was raised in the user trials. When a remove message is received by the Q-PIT whose identifier matches that of a search item, rather than a grouping item, the Q-PIT can delete that item from memory thereby removing it from subsequent regenerations of the Q-Space.

Finally in this walkthrough we come to the deletion of a Q-Space. Once a query has been finished with, a client can remove it by sending a `quit_query` message (figure 18). The Q-Space to be removed is identified by the `query_id` value. When this message is received by the Q-PIT it unloads the query associated with the given `query_id` from memory.

```
Client:    quit_query query_id=555
Client:    Bye
```

Figure 18: Removing a Q-Space and disconnecting.

The second message in figure 18 is sent by the client when they want to disconnect from Q-PIT altogether. This causes the Q-PIT to drop the socket connection, clean up and free any memory that it has been allocated and then unload itself from the Q-PIT server.

Implementation Issues

All of the implementation issues that arose from this new architecture came from the Q-PIT itself. The Data Extractor did not need much recoding to get it to read as well as write to the database cache. The previous architecture outlined at the start of this document was all written in Java, with the exception of the JDI or Jive code that interfaced with DIVE. In decoupling Q-PIT from the DIVE visualiser, according to our stated goal of separating data processing from visualisation, we had to re-implement a lot of the functionality that was originally handled by DIVE/Jive datatypes. Having done this, we then tried to make the Q-PIT more generic so as to be able to add new data sources without having to recompile the code each time. This involved scrapping the 50 character array attribute fields, which caused truncation of long titles, and replacing them with more flexible datatypes, such as Java String objects. Data arrays were replaced with List and Array implementations using the Java Collections API [Collections]. The classes in this API allow thread-safe access to the data that they contain and proved

particularly useful for creating and managing messages to and from the socket interface. This formed the basis for the first real implementation, but after initial testing it was found that its performance had dropped to less than that of the original demonstrator. This was due to the overhead of instantiating and accessing the Java objects. The second implementation of the new architecture performed far better than the first. Critical sections of Q-PIT processing code were optimised as much as possible through the removal of the Java Collections API code and the reintroduction of arrays. Strings were still being used to hold attribute data for each record and it was felt that, in the similarity calculations, the Java String compare function was not as fast as it might be. To coax the best performance out of the Q-PIT code, the implementation of the Q-PIT processing has been moved from Java to C, but Q-PIT socket interfaces still remain in Java. The Java-C communication and control takes place across the JNI, Java Native Interface. It is very much the case that the Java controls the executing of C functions in this architecture. Only in extremely process-intensive functions, such as the calculation of the minimum spanning tree, does C code invoke Java methods so it can issue 'progress bar information' updates to the client. The C code is compiled into a .DLL (on Windows platforms) or .SO (on UNIX) file and instances of the compiled code can be created by one or more Q-PITs. This is a crucial feature for the service that Q-PIT offers; in previous versions of the demonstrator, only one Q-Space could be produced at a time because there where many static variables and functions within the code. By removing this static code it has been possible to create a Q-Space generator that can handle multiple Q-Spaces at once. The Q-PIT has used this to good effect by turning itself into a Q-Space server – allowing multiple clients that conform to the Q-space API to connect request a Q-Space, whilst maintaining separation between data capture, processing and eventual visualisation.

When a keyword is received by Q-PIT a new instance of the C-based Q-PIT processor is created and initialised with default values for factor and exactness, together with the default schema and mapping for the data source being used. As the results are returned from the Data Extractor, Q-PIT loads them one at a time into the Q-PIT processor. The results are only stored in the C-code, they are not replicated on the Java side. Once all the results have been added the Q-PIT invokes the Q-PIT processing steps, one at a time. All of these steps are executed in the C code by calls made from the Java side of the Q-PIT. Once all of these steps have been completed the Java side of the generator extracts the representation from the C code and sends it via the socket interface to the visualiser.

Conclusions

In this chapter we have outlined the architecture for the original eSCAPE Library demonstrator, the changes that have been suggested by user studies of the

demonstrator and the architecture of a new Q-PIT system. The best way to address these issues was considered to be defining clear boundaries between data capture, data processing and visualisation. In this chapter only the changes in data capture and processing were reported. Actual visualisers that use the new architecture are reported later on within this deliverable document. The fact that the information being visualised was abstract, lent itself well to this separation of roles. When calculating an abstract Q-Space, how it is presented is not important to the calculations being performed to generate it. Overall we have lessened the impact of change on the system as a whole if any one part of it changes, and allowed change to take place, both in the data sources that can be used and the types of visualisers that can use them.

In terms of data collection (the Data Extractor), we have made the management of the cache database a lot simpler. There is no longer any need to register each potential Q-PIT host or end-user with the database management system. The Data Extractor can be located anywhere on the network, it does not have to run on the same host as the Q-PIT. Indeed it is a distinct advantage to run the Data Extractor in just location because of the volatile nature of the data source's presentation. As the data source is owned by a third party and not under control of the Data Extractor it is likely to change at any time. How the data is represented by the data source itself effects the parsing mechanism of the Data Extractor. When a change occurs it is easier to change a single Data Extractor rather than have to contact several. A future alternative to the current Data Extractor architecture would be one that is based on the Sherlock search system [Sherlock] developed by Apple for use in their operating system. The Sherlock system also has to parse web pages of results from a given data source, but the rules for this parsing are also online. When the representation of the results data changes, the parsing rules file for that data also changes. Parsing rules are downloaded by a user's client software every three days or so and stored locally. Using a system that can read Sherlock parsing rules files, would allow us to distribute the data capture activities onto the Q-PIT host machine. The drawback of this action would be that each host would require a caching system (the act of parsing does not go away, it is only shifted to the host). Running a database on each host would be inefficient so an alternative solution, such as writing the results to a local file, would have to be used.

Using one Data Extractor as a gateway to the data source also makes the cache database more effective, increasing the likelihood of a keyword being in the cache than if there were multiple caches separated across many sites.

The Q-PIT itself is host independent, having been separated from the visualiser, both in terms of location and the tasks that it must perform, which is particularly useful when considering the processing requirements of generating a Q-Space. The developer of a visualiser, once they have established a connection to the Q-PIT is free to experiment with the representations that can be used to display the Q-Space and the parameters that accompany its generation. Many different types of client can now have access to Q-Spaces with some requiring more processing to be performed than others. A PDA or mobile phone would not

be able to handle the generation of the Q-Space on its own, which is why the autonomous Q-PIT is so important. A 3D visualiser can allow end-user control of the positions of Q-Spaces within the abstract landscape, because the Q-PIT does not need to know the exact position in space. When a Q-Space is moved by an end-user there is no need to stream back these movements to the Q-PIT. Similarly the streaming of updates from the Q-PIT to the visualiser has been reduced. The force displacement process for example does not need to update a Q-Space for every iteration of its algorithm; this would produce a large volume of messages being sent to the visualiser. Trying to show the progress through the Q-Space generation process by animating the Q-Space itself does not work for other types of visualiser which do not or cannot display a Q-Space as a two or three dimensional object. It is far simpler and more effective to send status information to the visualiser, giving it the freedom to choose how to represent the progress throughout the generation process.

Overall the changes in the architecture that we have outlined in this chapter have allowed the system to develop from an electronic landscape based entirely on one visualisation platform into a more generic information *service*, where multiple data sources can be accessed freely by heterogeneous visualisation clients. In terms of the seven requirements that we set out earlier, for a new architecture, we have seen that the *separation of presentation from data processing* has allowed, for the first time, new types of visualiser access to a Q-Space. In addition it has allowed existing visualisers the freedom to experiment with the actual visualisation itself. The *separation of data capture from data processing* means that new data sources can be added to the system with minimum effort and impact on other architectural components. The internal structure of the Q-PIT has been changed to allow *scalability*, both in the number of search items that can be handled and data sources that can be accessed. Finally, *browsing by elimination* and for the OPAC system, *queries on classmarks* are also supported, answering user concerns that were identified during the previous set of user trials.

References

[Benedikt] Michael Benedikt, “Cyberspace: Some Proposals”, Cyberspace: First Steps. pp 273-302. MIT Press (1991)

[Berners-Lee] Tim Berners-Lee, “Hypertext Markup Language 2” RFC 1866, Section 8.2.1 (November 1995).

[Collections] The Java Collections Framework and API. http://www.java.sun.com/products/jdk/1.2/docs/guide/collections/index.html

[Crabtree] Andy Crabtree. “Understanding Searching as a Socially Organised Achievement in the Library” (Chapter 1). Escape Deliverable 4.1. (1999) Esprit Long Term Research Project 25377, ISBN 1 86220 079 3.

[Dive] Carlsson, C., Hagsand, O. 'DIVE: A Multi-user Virtual Reality System', in Proc. IEEE VRAIS, (September 1993), pp 394-400.

[JDBC] JDBC Data Access API.
http://www.java.sun.com/products/jdbc/index.html

[Mariani] John Mariani. "Q-PIT: Evolving an Information Terrain into an e-scape" (Chapter 7). Escape Deliverable 3.1. (1998) Esprit Long Term Research Project 25377, ISBN 1 86220 053 X.

[Mariani and Colebourne] John Mariani and Andy Colebourne. Q-PIT and Dataclouds: the generative algorithms." (Chapter 3). Escape Deliverable 5.1. (1999) Esprit Long Term Research Project 25377, ISBN 1 86220 081 5.

[Mariani and Crabtree] John Mariani and Andy Crabtree. "The Abstract eSCAPE Revisited" (Chapter 6). Escape Deliverable 4.3. (2000) Esprit Long Term Research Project 25377. ISBN 1 86220 098 X.

[Mariani et al.] John Mariani, Jonathan Trevor, Mårten Stenius, Detlev Schwabe. "The abstract electronic landscape demonstrator"(Chapter 5). Escape Deliverable 4.1. (1999) Esprit Long Term Research Project 25377, ISBN 1 86220 079 3.

[Mariani et.al.2] John Mariani, Jonathan Trevor Lennart Fahlen, Mårten Stenius, Annika Blunk "Concluding remarks and Future Possibilities" (Chapter 8). Escape Deliverable 4.1. (1999) Esprit Long Term Research Project 25377, ISBN 1 86220 079 3.

[Mariani and Trevor] John Mariani and Jonathan Trevor. "The Demonstrator in use" (Chapter 6). Escape Deliverable 4.1. (1999) Esprit Long Term Research Project 25377, ISBN 1 86220 079 3.

[mySQL] http://www.mysql.com

[Schwabe] Detlev Schwabe, "IO-dencies – Summarised technical description", eSCAPE Workshop 3, Lancaster, January 1998, deliverable input.

[Sherlock] http://www.apple.com/sherlock/index.html

[Stenius and Trevor] Mårten Stenius and Jonathan Trevor. "The Java-Dive interface." (Chapter 4). Escape Deliverable 5.1. (1999) Esprit Long Term Research Project 25377, ISBN 1 86220 081 5.

[Weiss] Mark Allen Weiss. "Data structures and algorithms in Ada", Addison-Wesley (1993). ISBN 0-8053-9055-3

Chapter 11: Alternative Interfaces

Part of the remit of the eSCAPE project was to investigate and provide alternative interfaces to the information used to generate the Q-space library demonstrator visualisation.

The following three subchapters describe three alternative approaches taken during the final year of the project. They cover the use of innovative hardware platforms, an installation environment, and an increasingly common use of mobile phone technology for accessing and displaying data. These alternative approaches have been made possible due to the revised architecture described in Chapter 10, which has resulted in a so-called "Q-PIT engine", providing a clean interface to the retrieval and generative functions of the Q-PIT.

Chapter 11a: The Tiltboard

This chapter presents a purpose built interactive device that combines a tilt sensitive accelerometer and touch screen to provide a tiltboard capable of displaying the Q-space. The chapter reflects on the device in use and the developments of a specialized interface to the Q-Space that exploits this.

Chapter 11b: The Pond

This chapter describes a more radical interface to the Q-pit engine that exploits a broad collection of interactive devices to allow access to online information. The Pond allows access to online information (in this case from Amazon) and describes a presentation of information that exploits a range of interactive devices.

Chapter 11c: The WAP interface

This final interface shows a minimal interface to the Q-space by providing a dynamic web and WAP based interface. This interface allows mobile access to range of different information sources and presents the groupings calculated using the similarity matrix.

Chapter 11a: Alternative Interfaces -The Tiltboard

Anders Wallberg, Andy Crabtree
SICS, Lancaster University

Year two of the eSCAPE project saw the development of a navigation 'vehicle' enabling users to interact with the abstract electronic landscape in a quick and easy fashion. That vehicle had two modes of interaction – 'point and click' and 'browse'. Users moved between these modes by pressing the space bar. In the first mode, objects were selected by clicking on them with the mouse cursor. This centred the object within the display and information about that object appeared on screen. The second method of navigating - the 'browse' mode - let the user orbit around a selected 'focus' object by moving the mouse. Keeping the view fixed to a central 'focus' object, prevented users from getting lost or looking away from the data. Our purpose in this section is to present an interface prototype – the *Tiltboard* - developed in an attempt to take this type of interaction away from the desktop into a physically based user interface that provides an engaging experience. The Tiltboard is composed of a touch sensitive LCD screen and an attached accelerometer[1]. The accelerometer lets us use the tilt of the device itself as input through measuring orientation relative gravity. By giving the user an easy way of viewing the search result molecule he or she can quickly gain a feeling for the structure of the search result and identify areas of interests.

1 The tiltboard is a low cost version of a growing number of devices that combine a display and a tilt device. For example see http://www.virtualresearch.com/products/win_vr.html

Figure 1: Using the Tiltboard

We describe the technical features of the interface, its functionality, 'hands on' experiences of users, and improvements implemented in light of user trials.

Technical description

The largest part of the Tiltboard is the touch sensitive LCD screen. We have used two different versions; the FPT-1212O and FPT-1503O open frame modules, both from Caltron Industries, Inc. The first one is a 12.1" SVGA screen with a weight of 2kg and the second is a 15" XGA screen weighing roughly 4kg. By accepting the heavier screen in our prototype setup we gained better resolution but more important a viewing angle of 160 degrees, both horizontally and vertically.

To measure the tilt we used the ADXL202EB from Analog Devices, which is a low cost ±2g dual axis accelerometer. The sensor is mounted so the axes are parallel to the screen plane, which allows us to measure the screen's orientation relative to gravity. On the evaluation board is a PIC16C63 microcontroller, which converts the sensor's pulsewidth modulated signals to RS-232. The serial communication is done at a speed of 38400 baud. We use 15 samples initially at startup to calibrate the tilt sensor. The number of samples used for each update is user configurable, but most often 3 averaged samples seem to produce stable enough values for every update.

The Tiltboard device driver has been implemented as a plugin to the distributed virtual environment software platform, DIVE.[1] The plugin also implements the features of the navigation plugin developed during year two and together with a search client plugin form a searching and browsing client in the generalised Q-Pit engine (data extractor, server, and client) described in chapter 10.

Functionality

The user can, through the touch screen, click on search result objects (see figure 2.) and use them as 'focus' objects, spinning around them at a distance by tilting the Tiltboard. Tilting the screen left and right makes the molecule rotate around the 'north-south pole' axis of the 'focus' object. When the user tilts the board away or towards him/herself the molecule will rotate around the 'east-west' axis of the 'focus' object. As there is no special 'up direction' in the abstract electronic landscape these axes depend on the orientation of the user, i.e. the Tiltboard. However, the rotation of the search result molecule as seen from the viewer is actually the viewer orbiting around the 'focus' object in the molecule at some distance. That is what is seen from another user browsing the same molecule through the multi-user capabilities of DIVE.

After preliminary user trials we had to add keys which allowed the user to invert the molecule rotation direction along both axes. Some users wanted the molecule to rotate in the same direction as a ball placed upon the Tiltboard screen surface would, while others wanted the opposite behaviour where the Tiltboard acted in a way similar to a fighter aircraft joystick. The user can also adjust the sensitivity of the Tiltboard, i.e. the rotation speed of the objects as a function of screen tilt. Additionally, the user is able to set the angle of neutral reading, i.e. the angle of the Tiltboard where things are not moving. This allows users to use the Tiltboard in a horizontal position, 45 degrees, or some other angle they find comfortable. We also added a 'stay still area' around the angle of neutral reading to allow for some unintended movement of the Tiltboard.

[1] Frécon, E. & Stenius, M. (1998) DIVE: A scaleable network architecture for distributed virtual environments, *Distributed Systems Engineering Journal* (Special Issue on Distributed Virtual Environments), vol. 5, p. 91-100.

Hagsand, O. (1996) Interactive MultiUser VEs in the DIVE System, *IEEE Multimedia Magazine*, vol. 3 (1), p. 30-39.

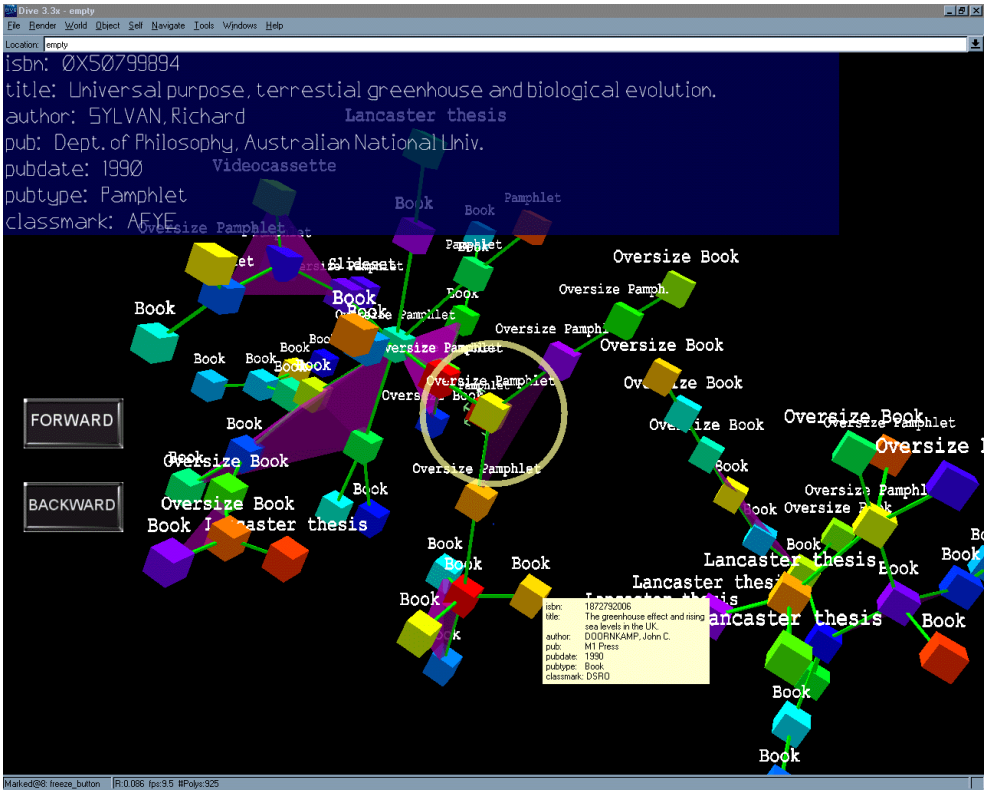


Figure 2: Browsing a search result space

Figure 2 shows an example of what can be seen in the Tiltboard when used with the library demonstrator as a database structure browser. In the centre is the selector circle. One object in this circle will have its data displayed on the HUD, the large rectangle seen in the upper part of the image. The same type of data can at the same time be shown for another object in a popup window, seen in figure 2, as a bright rectangle in the bottom half of the image. To the left, two virtual buttons, ‘forward’ and ‘backward’, can be seen. They are placed on the left side to be used by the left hand thumb during Tiltboard browsing but can easily be moved to support another way of holding the Tiltboard. The ‘forward’ and ‘backward’ buttons increase or decrease the distance to the ‘focus’ object, acting as a zoom

Getting ‘hands on’ the prototype – the outcomes of user experiences

The Tiltboard was confronted by real users during the annual Industry Day at SICS (13/04/00). The Industry Day is an event where companies working together with SICS on projects, can gain a better view of what is happening in other areas at SICS. The event also serves as a good starting point for discussions with companies that might be interested in collaboration. The Industry Day contains both seminars and demonstrations. The seminars, hosted by SICS employees, cover active research at SICS as well as current trends. Demonstrations were

displayed throughout the premises, enabling visitors to gain a detailed view of ideas, systems, and solutions being developed. The Industry Day attracts people from different areas; the majority being from companies in the computer or telecom industries, while some are from the business media, venture capitalist firms or members of the public. The vast majority of such visitors have a working knowledge of computers and what they may be used for; at least when it comes to standard uses and interfaces. Naturally, visitors are not so familiar with the potentials and constraints of less traditional uses and interfaces.

The Industry Day exposed the Tiltboard to a wide variety of users in serendipitous circumstances. User's motives for interacting with the interface varied widely, ranging from the curious to intensely interested. In all cases, however, the 'focus' of interactions was the interface itself and not the accomplishment of searching's work as in the user trials of the library demonstrator. In *this* setting the library demonstrator was little more than a passing interest, which served only to enable users to experiment with the interface. The nature of the day, particularly the flow of people through the various displays, effectively occluded further tests of the library demonstrator, being neither the time nor the place for conducting extensive user trials. Nonetheless, the day did lend itself to testing the practical efficacy of the Tiltboard as a ubiquitous interaction device and 'hands on' experiences were very productive. Some thirty to forty users confronted the interface over a period of some five hours. Users worked alone and gathered round in small groups of two or three watching others and subsequently assuming use as with the artworks at other exhibitions.[1] The overall opinion of users was that the Tiltboard was, in principle, a very good device for interacting with 3D environments. A small but significant number of improvements could be made to make the Tiltboard more usable to harness its potential however.

In interacting with the Tiltboard, the 3D environment kept 'spinning' around in response to slight movements of the interface. Users suggested that some kind of 'freeze' mechanism would be desirable, enabling them to hold the particular parts of the 3D space they were interested in, in a static position for easy viewing of content. The Tiltboard offered only one mode of navigating the 3D space. Movement of the interface resulted in movement of the molecular chain which users could zoom into and out of and select objects for viewing therein. Users suggested that a second mode of navigation would be useful. In this new mode, the molecular chain would remain static and the interface would be used to 'drive' around the molecular landscape. Naturally, any such implementation should enable users to switch between the two modes easily. In order to drive around the landscape in this new mode of navigation, 'backwards' and 'forwards' touch

[1] For details of the socially organised work implicated in 'becoming a user' of a novel interface see Crabtree, A., Hughes, J.A., Rodden, T. (1999) *Towards a Common Methodology* (eds. Crabtree, A., Hughes, J.A., Rodden, T.), eSCAPE Deliverable 4.0, Esprit Long Term Research Project 25377, ISBN 1 86220 078 5.

buttons also need to be added to the interface. To prevent users ‘getting lost’ in the second mode of navigation, a touch button should be added that returns users to the last point of contact with landscape.

The Industry Day proved to be a valuable exercise in gathering user input into the design process, enabling the designers to confront the technology with an unbiased and varied assembly of users. The outcomes of "hands on" user experiences after using the Tiltboard have shaped subsequent development and are listed below:

1. A ‘freeze’ device enabling users to stop the landscape from ‘spinning’ is required.
2. An alternate mode of navigation that enables users to ‘drive’ around the molecular landscape should be implemented.
3. ‘Backwards’ and ‘forwards’ touch buttons need to be added to the interface to support the second mode of navigation.
4. A touch button should be added to the interface that returns users to the last point of contact with the landscape in order to prevent users ‘getting lost’ in the second mode of navigation.
5. Users should be able to switch between the two navigation modes easily.

Identifying the need for a ‘freeze’ mechanism is a relatively trivial outcome of the user trial. Identifying a need for a second mode of navigation – an unrestrained mode of 3D navigation – on the other hand, is a major outcome of ‘hands on’ experience.

Improvements implemented following user trial

Guided by the outcomes of the Industry Day, improvements to the Tiltboard interface have been made. Several new virtual buttons have been added to handle the new features as can be seen in figure 3. Below we consider each of these features in more detail:

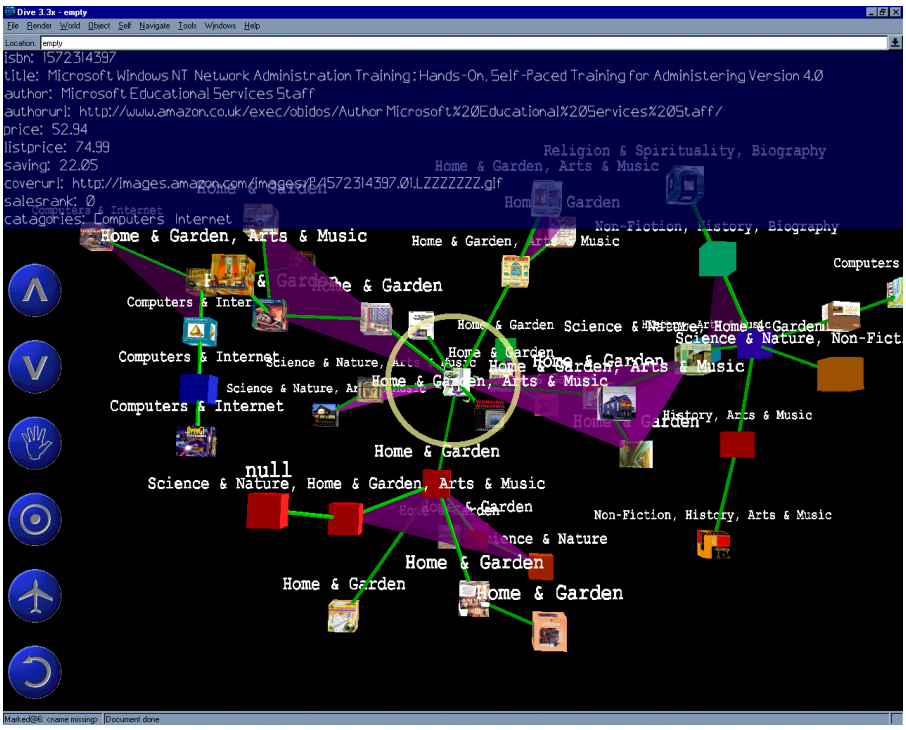


Figure 3: Screenshot from the Tiltboard with added buttons. The objects, links, and groups visible comes from three different searches in amazon.co.uk on the keywords Java, OpenGL, and Windows.

1. FREEZE MECHANISM

When using the Tiltboard to orbit around 'focus' objects the orbit speed is decided by the difference in angle compared to the angle of neutral reading. A small angular difference will produce a slow orbiting speed and a larger angular difference will result in a faster orbit speed. To be able to stop orbiting around the 'focus' object and keep the viewpoint still, one would have to gradually reduce the angle towards the angle of neutral reading and keep the Tiltboard still in that position. The fact that the tilt-sensor values are constantly used to update the view also made it difficult for the user to keep a certain viewing angle. An obvious and simple solution that was suggested several times during the user trial, is to use a 'freeze' button. When the 'freeze' button is pressed the tilt-sensor values will not be read, causing tilting of the Tiltboard to have no effect. Pressing the 'freeze' button again enables the tilt-sensor letting the user navigate with the Tiltboard. Technically it was easy to add the 'freeze' button, as it was just a matter of duplicating a feature that already existed on the keyboard, i.e. pressing the space bar. The 'freeze' button (with an image of a hand on it) can be seen in figure 3. directly below the 'forward' and 'backward' (or zoom) buttons, two buttons that also have keyboard counterparts.

2. ALTERNATE MODE OF NAVIGATION

The biggest difference in the Tiltboard interface that came out of the user trial is the addition of a second mode of navigation. In this ‘free flying’ mode the user is not limited to looking at a ‘focus’ object as in the ‘orbit’ mode but is free to move around. Tilting the Tiltboard in ‘free flying’ mode rotates the user accordingly, i.e. you control the direction you want to go. In ‘orbit’ mode, as explained earlier, the user is rotating *relative* to a ‘focus’ object, i.e. using the position of the ‘focus’ object as centre point for the rotation. Similar to the ‘orbit’ mode, it is the angle of the Tiltboard compared to the angle of neutral reading that sets the rotation speed. Tilting the screen to the left makes you start to rotate to the left, tilting it down makes you look down, etc. However, in this new mode the angle of the screen only dictates the direction the user will move. The movement itself is controlled by a ‘forward’ and ‘backward’ button.

3. NAVIGATION BUTTONS

The same ‘forward’ and ‘backward’ buttons used in ‘orbit’ mode to zoom in or away from a ‘focus’ object are used to control user transportation in ‘free flying’ mode. This is not done in an effort only to save screen space or to reduce the number of touch buttons keeping the interface simpler. The behaviour of the buttons in both modes are the same, moving the user’s viewpoint backwards and forwards. This fact made implementing them to support the second mode of navigation very easy.

4. RETURN BUTTON

A built in feature of the ‘orbit’ mode is that users never get lost. This is because they always have a ‘focus’ object in the centre of the screen. However, users navigating in ‘free flying’ mode can easily get lost. In response to the ideas that came out of the user trial we have added a way for the user to quickly get back to the last point of contact with the data, i.e. getting back to the last ‘focus’ object. Returning to the last ‘focus’ object is done by pressing the return button seen in the bottom of the button column to the left in figure 3. This rotates the user to look at the ‘focus’ object but keeps ‘free flying’ mode enabled.

5. SWITCHING BETWEEN MODES

With the addition of a second mode of navigation comes the need of a simple way to switch between modes. We have implemented this by adding two more buttons to what has become a button column with six buttons that may be controlled by the left hand thumb. One mode switch button would have been sufficient letting the user switch to the other mode. However, as the number of modes was limited we opted to display the mode choices and highlight the button associated with the active mode. The ‘free flying’ mode button can be seen in figure 3 covered by an image of an aeroplane, and the orbit button just above that.

Conclusion

In this part we have reported on the work done on an advanced interface to an electronic landscape, namely the Tiltboard. The Tiltboard is a tilt-sensitive interaction prototype developed as part of an abstract electronic landscape although it may be deployed in conjunction with physical eSCAPEs. The Tiltboard has been interfaced to DIVE and evaluated through user trials. The outcomes of 'hands on' user experiences have shaped a number of significant improvements to the interface, particular the implementation of a 'true' mode of 3D navigation which is both simple and user friendly.

The Tiltboard is representative of the development of novel interactive devices as part of the eSCAPE project. The development of these new classes of interactive device opens up new interaction possibilities. In the following subchapters we explore how a range of these devices can be used in tandem to provide an alternative interactive experience.

Chapter 11b: Alternative Interfaces - The Pond

Lennart E. Fahlén, Anders Wallberg, Jan Humble,
Jonas Söderberg, Olov Ståhl
SICS

The Pond is an effort to bring together a number of ideas from different fields and also to further develop the concepts, technologies and design ideas central to eSCAPE projects, such as the Blob, the Web Planetarium, and the Library Demonstrator. The Pond is a multi user horizontally projected system for access and manipulation of fluid information elements from the Internet (e.g. the WWW) as well as for communication and collaboration between users being present both remotely and physically. The choice of a Pond metaphor is deliberate in that it seeks to evoke the general concepts associated with evolutionary ecosystems and techniques where complex assemblies of users, electronic agents, interface and communication resources are dynamically built and destroyed according to need.

The Pond and the Library Demonstrator

In the Pond design we try to move away from the conventional ideas of desktop metaphors and the established notions of how information systems are accessed and manipulated. Instead, concepts from areas such as bio-ecosystems, fishing, fluid mechanics and roulette wheels and jukeboxes have been our inspiration.

The pond demonstrator is a much simpler information browser and visualizer than the current Escape Library demonstrator. For instance, search criteria are indirectly derived by selection and association links are much cruder and to a great extent not user specified. On a more detailed level the pond instead of using hulls to display groups of object as in the Library Demonstrator, explores flocking behaviour as an alternative approach to convey group belongings.

The Pond application focuses more on user interface issues, social browsing and direct interaction compared to the batch job metaphor of the library demonstrator. It is our hope to eventually be able to bring the two approaches closer together but for now the pond table is an experiment in alternative user interfaces.

The content

The source of information we have chosen for this application is the electronic online music/CD store of Amazon.com. There are several reasons for this choice. The database is easily accessible and is formatted in a uniform and simple way.

In addition, nearly all of the CDs available have one or more 30-second sample audio clips associated with them in a standard and easy to play format (RealAudio). Maybe most interestingly Amazon employs a recommendation system whereby each CD has links to other CDs people bought as well i.e. “Customers who bought this title also bought…”, and there is a list of (typically) 5 links to related music items. Finally the concept of music selection and listening is an established and well-understood process by many laypersons.

The set-up

The escape demonstrator uses a non-stereo desk projection with the projection coming from underneath via a mirror placed under the desk. There is also an overhead structure containing video tracking (see below) and sound projection equipment. Three sides of the tabletop are lined with switches hidden in pads being triggered by users leaning on the desk or by simply pressing a pad. In the centre of the Pond set-up is a touch enabled horizontal display. This display, which has the form factor of a large table, presents the view into the pond. Users can sit or stand around the display when working with the Pond.

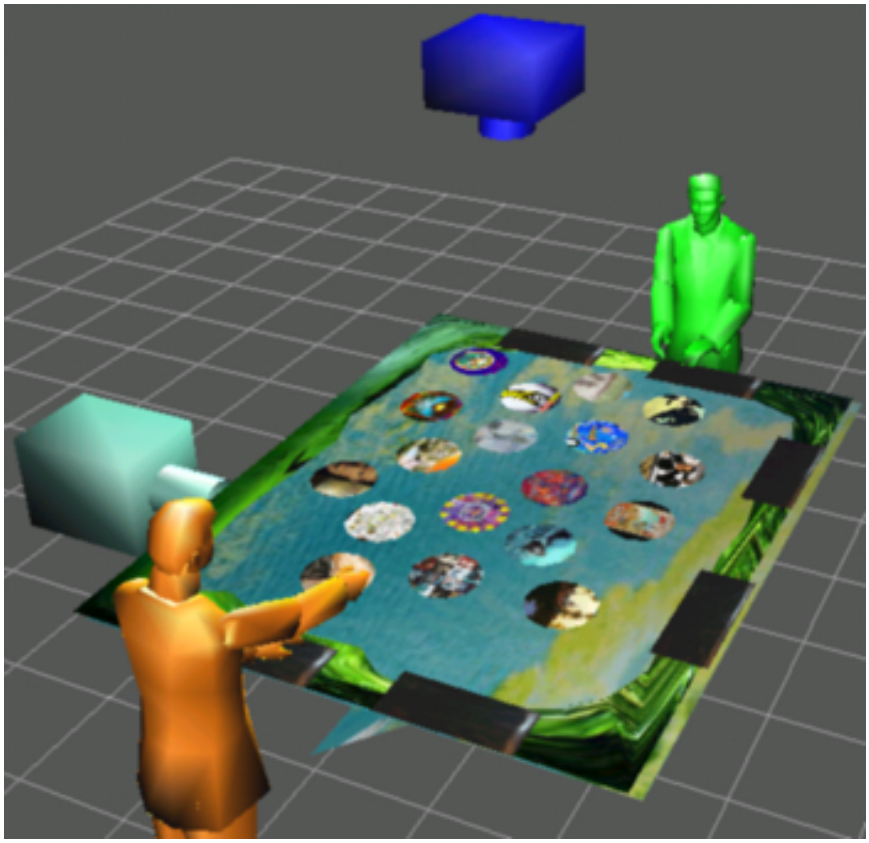


Figure 1: A Pond model

Three dedicated workplaces exists around the central display area, each equipped with pressure sensitive surfaces. A pond table is intended to be operated by one to six users simultaneously and the limitation to three users is just to make the current demonstrator practical. There are no keyboards or mice input devices, instead user awareness and input is handled via gestures, pressure pads and tag operations.

The ecosystem

The Pond world tries to convey the feeling of a dynamic, changing and living environment while at the same time providing a rich set of possible feedback mechanisms. The always moving pond environment can be used in interesting ways to, for instance, convey system information, e.g. a less turbulent pond surface could indicate a slow connection to the rest of the world.

The main type of pond creature in the present application is the CD-creature. It looks like a flying saucer or a flattened out blob with its skin texture taken from the CD cover it represents. The CD creature responds to touch and speed of movement and uses interpolators to drive its (fish) animations.

Behaviour and interactions

A user is given a token (i.e. RF tag see below) preloaded with links to some music. The user moves up to the pond table and puts down the token on the tag reader. After a few seconds, a music icon rendered with information from the album cover will appear in the pond and the preloaded music will start to play. Soon thereafter related music selections will begin to surface and swim around. There is a continuous supply of new items introduced as if coming from a stream upwelling in the centre of the pond. Items can also be introduced by a user with a set of music style tags. As each item reaches the surface a few seconds of a song from the CD that item represents is played. The item then slowly starts to float round the pond. If a user selects an item when it floats by him the item will rise up towards the surface and again start to play one of the available audio clips. At the same time other related items in the pond will start to move towards the selected item. If the user decides that the item wasn't really appealing and deselects the item it will sink down into the pond and disappear. Furthermore, the flocking behaviour of related items will stop and the circling around the pond of the other items will continue. If, on the other hand, the user continues the selection the item will move to a spot in front of the user and position itself there and the related flocking items will stay in the vicinity and wait to be selected as well. If a selection does not occur within a specified timeframe those items will sink into the depths of the pond and disappear. The items having been collected in the above way will constitute a school of aquatic creatures flocking around the user. This group or flock can then in a very simple manner be "offloaded" to a tag of the users choice. If the user at a later point in time puts the tag back on the

appropriate tag reader the members of the group contained in the tag will be reintroduced one by one into the pond and the whole process starts all over.

Search refinement is done by the system detecting user actions that convey interest in an item such as playing a large set of the available clips. With the help of the related item information on the Amazon.com database, the system can insert new items that have a high chance of being interesting to the user as well. Combined with the fact that items considered uninteresting will sink and disappear, this has the effect of gradually refining the ecosystem to contain more and more interesting items. This procedure also allows for moving through the set of available items in that data source without burdening the user with endless dialogue windows and manual cleanup operations. So even when very specific selections are being used as "seed" material, the pond will at best have represented in it several general tastes, specific favourites and complete wildcards.

Pressure pads and RFID tags

To simplify the process of getting input from the pressure sensitive pads distributed around the Pond display area we have connected the cables from the pads to the internal wiring of the keyboard attached to the system driving the Pond. Putting pressure on a pad in this set-up will result in the generation of the corresponding letter on the keyboard. The Pond process will only have to monitor keyboard events. When a pad is pressed the Pond process will start listen to the RFID reader next to the pad to acquire the id of the token placed upon it.

Figure 2: Pressure sensitive pad

The tags used in the Pond are divided into two groups, the predefined search categories tags and the personal selection tags. The predefined search categories

tags are associated from the start to different categories in the subject being browsed. In the case of music the styles, this could for example be ‘Classical Baroque Symphonies’ or ‘Drum & Bass’. Putting one of these tags on a reader will populate the pond with search results in this category. The second type of tags, the personal selection tags, have no predefined associated data. These tags are used as markers to information gathered while Pond browsing. If the user puts a selection tag on the reader unit the information in the pond creatures he has caught will be saved and associated with that tag. Putting that tag on a reader unit at some later stage will bring back the information.

Figure 3: The Radio Frequency ID reader and a bunch of tags

Radio frequency identification (RFID) is a relatively new technology, first appearing in tracking and access applications in the 1980s. The use of RFID systems for commercial and scientific applications is gathering speed and the full potential of RFID technology has only just started to be realised. A RFID system consists of three main parts; a data carrier also called a tag or a transponder, an antenna and a reader unit. The antenna generates an electromagnetic field that energises a chip inside the tag. The electromagnetic field is modulated by the tag and the modulated signal is recovered by the antenna and sent to the reader unit for interpretation. The read-only tags have pre-programmed unique codes that are interpreted by the reader unit. For example, a 64-bit code could after interpretation be presented as a 10 digit hexadecimal code. The code in a read-only tag cannot be changed in any way.

The Sonic Interface

In the present version of the Pond, sound is used at two levels. The first level is the ambient environment. Since the metaphor is a "pond" or "pool" of information, the soundscape should consist of a family of wet, bubbling, splashing and whirling sounds, giving the impression of the information-units (the music records) ascending from an abyss of ooze or mud. The ambient sounds fades out when the chosen music-sample is played, and fades smoothly in when the music stops. The second level is the interface-sounds indicating user-presence, selecting/unselecting, clicking, dragging etc. These sounds do also originate from the above-mentioned family of pond sounds. The music samples associated with the Amazon.com database of CDs is a further level in terms of audio. These music clips are played "outside" (i.e. not spatially rendered) of the 3D pond environment using an adapted Real Audio player.

Real Time Video Tracking

A set-up composed of a back-projection table and a top-mounted standard camera together with a standard PC provides a tracking system option for interfacing with the pond environment, providing 1-3 users an interface relying on gestures. Robustness and a certain level of flexibility are imposed on the tracking system. Several alternative feasible approaches are currently available for the implementation itself. The basic concern is speed and response time, and thus we discount masking filters or any higher order image operation for the task. We are not concerned with orientation or identification of objects, only basic movements and gestures. One approach is the use of an infrared device to follow human limbs across the surface for selecting and moving pond objects. Almost any common camera can be adapted to capture infrared wavelengths, eliminating the need for segmentation from background. A blob detection algorithm is hereby useful to isolate certain elements and couple them together with movement, such as differentiating a still hand from a moving one. Since we have the information of the projected background we can also use image subtraction to separate the real components from the projected ones. The difficulty here is calibrating the background info to the resolution and image co-ordinates from the camera, and synchronising the two. However, scaling the projected space to conform to that of the camera might provide enough usability.

A less common approach is human skin tracking. The projected background is flat and artificial in nature. With an average camera, good results in skin tracking have been seen to work; but again, only for basic blob-like features. Expanding on the idea, users could wear or handle objects that are as distant in nature to the projected background as possible; but this is an ugly and less flexible solution. Once any such segmentation is accomplished it is relatively easy to interpret basic movement into gestures, provided that we don't opt for precision. Precision will require filtering algorithms that are computationally expensive and require

continuous calibration. Large-scale co-ordinates and relative orientation of movement are possible. Velocity is extracted provided the system accounts for frame drops.

Shoulder to shoulder collaboration

Most computer applications written today are single user applications. They are intended to run on a single workstation controlled by a single user. Systems that support group collaboration are traditionally run on multiple workstations communicating across a computer network. In addition to supporting collaboration between remote participants through peer-to-peer communication the Pond also supports some of the ideas in the Single Display Groupware (SDG) model[1]. SDG focuses on co-present collaboration with an input channel for each user and a single shared output channel. The shared output channel in the Pond consists of the visual and aural rendering of the Pond based on actions performed by all participants. The input channels are the pressure pads, the RF tags, the touch screen and the video tracking.

In the available version of the Pond, one application could be in a frequent party-situation, as a tool for a small group to discuss or debate music or to argue about their musical preferences.

Web connection

Similar to the library demonstrator the Pond is an interface to active online information. The interface between the Pond and the web is the data extractor. When a user issues a request by placing a category tag on a tag reader or expresses an interest in information related to a pond object being explored, the data extractor checks if the request has previously been made and the results stored in a MySQL cache database. If not, it will capture the information from the selected data source (in this case the CD collection at Amazon.com), parse the results and store it in the cache database. It then returns the identifier associated with the new results. The Pond process makes use of the data stored within the cache database and process it to generate the visualisations presented to the end-user.

JIVE

To control the pond interface we use Jive, the Java-DIVE native interface, developed during year two of the eSCAPE project as a way to let Java processes

[1] Stewart, J., Bederson, B., Druin, A. (1999) *Single Display Groupware: A Model for Co-present Collaboration*, CHI 99, p. 286-293.

become full members of DIVE[1] worlds. Jive is described in the second year eSCAPE deliverable[2]. The biggest change in Jive during this third year has centred on speeding up Jive calls that modify Dive entities.

Discussion

We have explored the concept and architecture of the Pond, an information ecosystem metaphor used as an interface to information on the Internet. There are several crucial issues to the design of especially multiuser interface artifacts being explored in this application such as the choice of interaction metaphor, the exploration of shoulder-to-shoulder collaboration, the "purpose" and role of the application itself and the adaptability and usefulness of the alternative technologies being explored.

Firstly, let us examine the actual nature of the pond table concept. In the Web Planetarium, users had to select items (web pages) by direct choice, i.e. by clicking on links in 3D space. There was no implicit introduction of new elements by the underlying system. In some applications this turned out to be a problem because users with no prior experience were too shy to discover what had to be done in order to introduce a new web page. The Library Demonstrator on the other hand is based on the concept of the user providing a set of (text) data in a way very similar to how a conventional database search is initiated and the number of items being searched are more or less infinite. The results of this initial search can then ,of course, be used for further searches in a more intuitive and user friendly way (i.e. by direct manipulation of 3D objects).

With the Cyber Pond we try to combine the above approaches with an emphasis on user interface details. There is a fixed number of entries that is potentially very large (the Amazon sales catalogue of music CD's) but there is no guarantee a specific item will be found or arrived at during any one session. The choices made are opportunistic and direct and very much dependent of what other users are currently interested in and the links being pursued when the underlying system introduces new items is very much ad hoc.

An inspiration for the present Pond, apart from the organic life, nature water etc. metaphors already discussed, is somewhat surprisingly the mechanical metaphor of the music jukebox and casino style gambling roulettes. The purpose of these contraptions is more or less obvious to people (users?) still the machines themselves carry an aura of intrigue and mystique. Many people find watching the

---

[1] Frécon, E. & Stenius, M. (1998) *DIVE: A scaleable network architecture for distributed virtual environments*, Distributed Systems Engineering Journal (Special Issue on Distributed Virtual Environments), vol. 5, p. 91-100.

Hagsand, O. (1996) Interactive MultiUser VEs in the DIVE System, *IEEE Multimedia Magazine*, vol. 3 (1), p. 30-39.

[2] Stenius, M., Trevor, J., (1999) *The Java-Dive Interface* (ed. Pettifer, S.), eSCAPE Deliverable 5.1, Chapter 4, Esprit Long Term Research Project 25377, ISBN 1 86220 078 5.

working of the mechanical mechanisms endlessly fascinating and the designs are seen as a work of art in themselves. In the jukebox example the mechanical noises it makes and associated delays in "processing" the coin and actually getting a record on to the turntable are seen as a natural part of the mechanism that brings a certain transparency and understanding of what the machine is currently doing.

In terms of how information is displayed to users as well as the actual form factor of the artefact one of the things we want to explore is the position of the display. In our experience a horizontal presentation surface where users can gather around have an invitational collaborative sense to it. In, for instance, a cinema or conventional overhead presentation there exists a large gulf between the presentation and the audience, there is a significant effort required to get involved with the activities surrounding the presentation. A horizontal display where users can stand or sit around offers easy affordance for participating even on such a lightweight level as simply pointing to details of the presentation.

A serious drawback with the large area touch screen technology we currently use is a limitation when it comes to detecting multiple simultaneous touches. Future true multi-user touch screens will alleviate this shortcoming. In the present set-up we try to work around the problem by only using the touch screen for functions that inherently are single user, i.e. for the triggering of music playback. It of course makes no sense to play several samples at the same time, the result would only be a meaningless cacophony of sound.

One lesser problem is that the use of a separate application for playing the music samples (i.e. Real audio playback) breaks the illusion of an integrated 3D environment. This problem has a simple remedy in that it will be pretty easy to integrate support for the media format (of ones choice) into the application software

The RF tag system being used in the set-up can play a set of different roles. One of the more fundamental questions is if the tags are to be viewed as holders of search criteria (e.g. "country music") tags or as containers of special music items (e.g. "the 1993 Seattle Grunge Christmas Album"). The first case implies more of search oriented application and given that there is no explicit character/keyboard input available to the user whereby more detailed search criteria can be defined (as in the Library Demonstrator). This function of the tags is probably not the most appropriate use.

We have opted to focus on the second usage of the tags whereby tags contain explicit references to pieces of music. The reasons for this choice are twofold. Firstly, the current version of the Pond Table is a demonstrator and the audience will probably not spend much time with the unit (15 minutes maybe). Secondly, keeping the handling of a music tag similar to how a music CD or cassette tape is used will reduce the feeling of unfamiliarity based on the new concepts used in the Pond. Behind the scenes there will of course still be an ongoing implicit search based on the Amazon recommendation database.

Given the nature of the tags and other functionality available in the pond it is also possible to imagine more interesting usages such as putting reflectors on top

of the tags and placing them in the pond to be tracked by the video tracking subsystem. In this case the tags could for example function as tracking targets for flocking pond items.

The pond is designed to be more fun and social than precise and analytical. Our working view of the current Cyber Pond application is that it is very similar to an old style jukebox, but with a twist (a cyber jukebox?). And just as a real world jukebox, a pond not being interacted with will have an interesting behaviour, it is a machine that is a pleasure to look at and listen to.

Finally, it would of course be possible to build a browser plug-in for a conventional desktop application but a lot of the fun and mystique of the pond artefact would be lost.

The Pond provides an example of a radical form of interface that can be used to access an on-line abstract electronic landscape. However, the notion of an abstract electronic landscape can also be experienced via more traditional and much less radical interfaces. We shall conclude our presentation of alternative interfaces in the following subchapter by considering the development of a web based interface and its presentation through a WAP interface.

Chapter 11c: Alternative Interfaces - WAP

Peter Phillips
Lancaster University

In chapter 10 we outlined the changes that have been made to the Library demonstrator to realise a generic Q-space engine. The architecture separates the generation of the Q-Space (an abstract object representing a search result) from the actual visualisation itself making it possible to introduce new visualisers with little cost. These interfaces include those seen in the CyberPond in the previous subchapter and the Tiltboard in chapter 11a. The revisions that have been made to the Abstract Electronic Landscape generator have opened up the possibility of using alternative representations of generated Q-Spaces. In this chapter we describe one such interface, based on the relatively new Wireless Application Protocol (WAP)[1] and in particular the use of the Wireless Mark-up Language (WML)[2].

WML – A brief description

WML can be thought of for phones as HTML can be for the World Wide Web, it allows the creation of content that conforms to a specific set of rules or specification and can be displayed in a browser. For HTML this browser can be a third party piece of software. For a mobile phone, a WML browser is usually a core component. An HTML file is usually displayed as a 'web page', i.e. a user scrolls up and down the page to view its content, there is no off-line or out of sight information. In WML, because the devices being used to display the content have such limited features (screen size, memory, colour palette, etc.), the content of a WML file is not displayed on one screen. A WML file is divided up into *cards*, with one card being displayed at once by the phone's WML browser. A WML file then, is actually a *deck* of *cards* rather than a page as in HTML. A card that is displayed in a browser can have text, which can be styled, images, input components such as text boxes or radio buttons, and hyperlinks. The way these items are described in a card is very similar to the corresponding descriptions in HTML (an <a> tag is used to represent a hyperlink for example.) For those with experience of writing HTML tools or content it is a relatively easy step to move to WML. This is also true for the serving of WML decks. HTTP can be used to distribute decks to clients who request them and CGI scripts can be written to

1 http://www.wapforum.org

2 http://www1.wapforum.org/tech/terms.asp?doc=SPEC-WML-19991104.pdf

dynamically generate desks to be served by web servers that have been modified to distribute WML content in addition to HTML.

Q-Spaces and WML

At first it appears that, by using WML, we are taking a backward step from our previous work; going from visualising a Q-Space in 3D to producing a text-based visualisation in WML. The original source of information for a Q-Space is usually a search facility on a web-site, such as Amazon.com. These sources already display the data as a web page or sequences of pages depending on the number of results returned by the search. The advantage of the three dimensional visualisation is that it allows a user to see the whole search space at once instead of 'paging' back and forth between short lists of search items. A feature that the WML visualisation has, which the original data source does not, is the additional processing of the search results performed by the Q-PIT, specifically the region identification. In reality though it can be said that we are not returning to the text-based visualisation of search results but are going beyond that, where the devices being used will find it difficult to display a single result at once, let alone a 'page' of results. The volume of information that can be displayed is an important factor in designing a phone-based interface.

In the remainder of this document we present the architecture and process of use of this alternative interface before highlighting some of the implementation issues that were faced and the lessons that can be learned from them.

The Architecture

Due to the limited facilities available on a typical WAP-enabled mobile phone it has become necessary to hold the Q-Space at an intermediate location, between the Q-PIT and the end-user's phone, so that parts of the Q-Space can be sent to the phone as it needs them. Therefore a requirement of this intermediary was that it should be accessible to a mobile phone, simply, that it should be able to deliver its content via HTTP and be able to communicate with the Q-PIT at the same time. The software chosen for this intermediary was based on Java Servlets[1], small pieces of Java code that can generate dynamic content for an HTTP response. This technology is usually employed to create HTML but should also be able to create WML. Architecturally, the new visualiser continues to adhere to the separation of content from presentation. This can be seen in the use of XML to encode and store a Q-Space once it has been returned from the Q-PIT that generated it. In deciding to use XML we had to choose a Java servlet engine that would be sympathetic to it. Eventually we chose Cocoon[2], part of the Apache project, as the web

1 http://www.java.sun.com/products/servlet/index.html

2 http://xml.apache.org/cocoon/index.html

server/servlet runner to be the intermediary between the phone and the Q-PIT. Cocoon is a publishing system based entirely on XML. The underlying design principle behind its development is the idea that data, its presentation and its management are separate issues. This allows a Q-Space to be encoded in XML with no need to hard code any display information within it. Other content creation technologies such as Active server Pages[1] or PHP[2] require data processing commands to be embedded in a template web page by a developer. This page is then processed every time an HTTP request is sent asking for it. This mixing of presentation and content goes against the ideas behind the revisions made to the Q-PIT architecture outlined earlier in this deliverable. Fortunately, Cocoon also abhors this mixing, making it suitable for our needs. The architectural components of this new visualiser are detailed below, after which we shall give a brief description of the process of creating a Q-Space for a mobile phone.

Components of the Alternative Visualiser

The architecture used to provide access to the Q-Space is built around a number of key components. These include:

Q-PIT: the Q-Space Generator

The WAP visualisation architecture is, in part, a client to the newly developed Q-PIT architecture, which has already been presented earlier in this deliverable document but a brief description of it is included here. A Q-PIT is a piece of self-contained software that is responsible for generating the Q-Space. It is invoked by opening a socket connection to a Q-PIT server and then sending a search term (keyword) that will be used to search a specified source of data. Modifications can be made to parameters that are used in the calculation of a Q-Space, causing the Q-PIT to regenerate the space.

Cocoon: a publishing system

The third party Cocoon publishing system is used as an intermediary between the end-user's device and the Q-PIT itself. Direct communication between a WAP-enabled device and a Q-PIT is not possible because of the limited specification of the device, therefore the use that we have made of the Cocoon system has been to connect a phone to a Q-PIT and store the complete Q-Space on behalf of the phone. The phone itself communicates with Cocoon across HTTP, whilst communication with the Q-PIT has been realised by incorporating an implementation of the Q-Space API (see chapter 10) into a Cocoon servlet. When a device connects to the Cocoon server the parameters passed in the URL are accessible to the servlet that is being invoked. These parameters can be used to

1 http://www.microsoft.com

2 http://www.php.net

control the execution of the servlet as well as contain data that the servlet can pass directly on to the Q-PIT.

Within Cocoon itself the Q-Space is stored as an XML file. There are no assumptions made about the visualisation of the Q-Space at this point, so the architectural components that we have described so far could just as easily be used to generate representations of a user's search using a different technique or file format. The conversion of an XML file into a WML representation of a Q-Space is achieved by using the Extensible Stylesheet Language (XSL)[1]. This is a very new technology; its specification is still being finalised, but fortunately Cocoon supports the latest features as it uses well-maintained Java packages. To translate one XML-based file into another XML-based file and XSL stylesheet is created that contains rules to apply to XML tags found in the first file in order to generate the second. The generated file does not have to be XML based, but the file that the stylesheet is applied to must be. In the case of this alternative interface, we have made the Q-Space file and XML-based file and created our own XSL stylesheet to convert that file into a series of WML files, which is also XML-based. Note that we produce a series of WML files (decks) from the single Q-Space XML document; as we have already pointed out, it is not possible for the end-user's device to hold a complete Q-Space at once. The Cocoon system allows the passing of HTTP request parameters to the XSL processor (the part of the software that takes the base XML document and applies the XSL document to it). We have used this mechanism to select which WML deck and should be generated and returned in response to a request.

WAP-enabled device

We have no development control over the phone or PDA that is being used to view the Q-Space. We can only assume that its WAP browser will be based on the WAP 1.1 standard and that, either directly or through a gateway of some sort, the device will be able to access the Internet.

1 http://www.w3.org/Style/XSL/

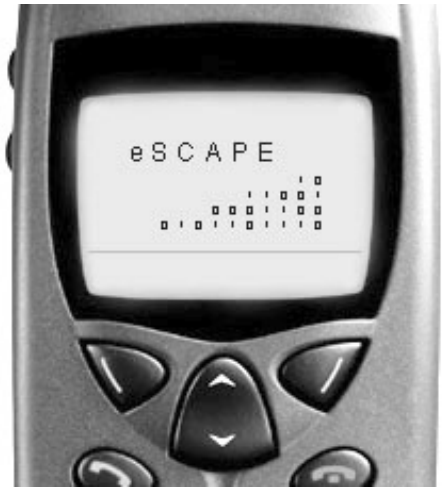


Figure 1: LCD Screen and Navigation Buttons

Figure 1 shows a typical WAP-enabled phone. The screen size on this particular model is black and white, measuring 130 pixels by 60 pixels. The three buttons, also known as keys, directly below the screen are known as 'navigation' keys. The left and right keys are 'soft keys' that can be used to select options that are displayed on the screen. The central button is in effect a rocker button and is used to move up and down a card displayed on the screen. Pressing the down arrow on the key moves the user's focus from the currently selected item, be it a hyperlink or input box, down to the next selectable item. If this next item is off the screen the card is scrolled to bring it into view. In situations where there are no selectable items on the screen but more information than can fit on to one screen, the card is scrolled down by almost a full screen's worth.

Accessing a Q-Space

The whole process of generating and accessing a Q-Space begins with a WAP-enabled client connecting to the Cocoon server using a known URL. This causes the Cocoon server to fire up a servlet, to which the client's HTTP request is channelled. The servlet that is invoked has the ability to communicate with a Q-PIT across a socket interface; the end-user's device does not access the Q-PIT directly. Upon receiving the client's initial request the first task of the servlet is to connect to a Q-PIT server, which spawns an instance of a Q-PIT that the servlet can communicate with. This connection takes place across a socket and the communication across it is based on the Q-Space API, which has already been detailed in chapter 10. Consequently we will not list the complete sequence of messages that are exchanged using the API, but we will point out some of the more important ones. Figure 1 shows the initial screen that is displayed on the user's device – a card containing only a single image. This card is then replaced after a fixed interval with the card shown in figure 2b.

```
Servlet:    list_sources
Q-PIT:      list_sources number=2
Q-PIT:      name=OPAC description=Library+OPAC
Q-PIT:      name=Amazon.com description=Amazon
```

Figure 2a: Getting a source list from a Q-PIT

Figure 2b: Selecting a data source

The card in figure 2b is prompting the user to select a site that they want to search. The text accompanying the radio buttons on the screen shows the data sources that the Q-PIT, which the servlet has connected to, knows about. The servlet as taken this information, given in response to a `list_sources` API message as shown in figure 2a, and added it as a card in a WML deck. Associated with each selectable item is a value. When the end-user has made their choice and used the left-hand soft key to choose 'OK', the value for the chosen item will be returned as a parameter in an HTTP request. Once the user has selected a site to search a final card is displayed that asks the user for the keyword to use (figure 3a).

Figure 3a: Keyword card

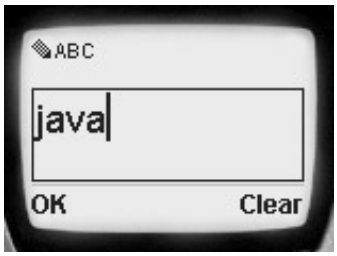


Figure 3b: Entering the keyword

Figure 3c: Card with keyword

Notice that the label corresponding to the left-hand soft key has changed from the previous card that the right-hand soft key also has a label. The right-hand soft key will take the user back to the previous card, where they can change the site that they wish to search. The left-hand soft key is used to invoke the currently selected item on the card. There are two selectable items on the card shown in figure 3a. The first is the input area, where a user's keyword is displayed. The inverting of the item shows the fact that it has the selection focus. Selecting this item changes the card to a built-in input card. The user uses the keyboard or keypad on the device to enter their keyword. Once they have done this they press the left-hand soft key, now labelled 'OK', which sends them back to the previous card (figure 3c). Notice that the user's keyword is now displayed in the input item on the card. To invoke the search the user moves the focus of selection, using the scroll key, down to the 'search' hyperlink, the second selectable item on the card.

Its appearance changes to indicate that pressing the left-hand soft key will select it; it is inverted and the left-hand soft key label has changed.

These three cards, 'logo', 'source selection' and 'keyword entry' constitute the whole deck that is returned by the servlet in response to the initial connection. Whilst the user is navigating within this deck there is no communication with the servlet. Parameter values that have been set in each card are stored within the device where they are accessible to every card within the deck. These parameters are used to create an HTTP GET request by adding them to the URL of the servlet. An example of such a URL is shown in figure 4.

http://148.88.152.249:8080/Q-pit/query.xml?source=opac&keyword=java

Figure 4: A URL containing name/value parameters

A URL such as this is sent as to the servlet, which reads the parameter values and sends two messages to the server. The first is a `select_source` message, which tells the Q-PIT the name of the site to search, the second is a `query` message, which contains the keyword to be used for the search. On receiving this final message the Q-PIT begins the generation of the Q-Space.

Keeping the User Informed

Whilst the Q-PIT is generating the Q-Space it is sending status messages back to the servlet that envoked it. The servlet has to interpret these messages and format them into WML that can then be sent on to the user's device. There is an architectural conflict at this juncture between the Q-PIT messaging system and HTTP. In the first case the communication channel (a socket) between the servlet and the Q-PIT is always open during the lifetime of the Q-Space and either party can send a message across it. Compare this with the connectionless, 'client pull' architecture of HTTP. The differences between these two communication techniques make it difficult to pass status information directly to the end-user's screen. The solution shown in this document is one of several potential solutions. Other solutions to a more generalised version of this problem will be discussed later in this document.

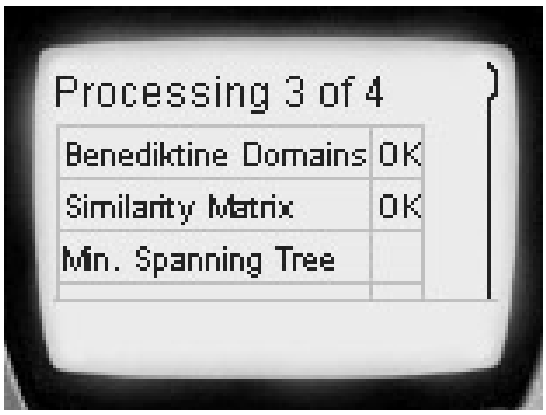


Figure 5: A card from the progress sequence

Figure 5 shows one card from a whole series that show the progress of the Q-Space generation process. There is no 'progress bar' or other animated indicator, simply a list of the functions that must be performed to complete the Q-Space and an 'OK' next to those that have been completed. Figure 5 can be interpreted as a snapshot of the Q-Space creation process; the 'Benediktine Domains' and 'Similarity Matrix' functions having already completed, the 'Minimum Spanning Tree' function in progress and the 'Region Identification' function yet to be started. (Note that the limitation of the screen size has meant that the 'Region Identification' label is not visible). The line of text at the top of the card informs the user of the number of functions to be processed and which function is currently active. A new card can be displayed only after a function has been completed. This is achieved by dividing putting each card in its own deck. When a card, such as the one in figure 5, is displayed, after a short wait (5 seconds) it issues a HTTP request for the next card in the series. If the function has not yet completed (for figure 5's case it would be the 'Minimum Spanning Tree' function) the servlet cannot return all of the content for the new card. So long as the delay is less than the device browser's connection timeout value, the function can complete and the servlet can return the complete card (in its deck). We want to make sure that the connection does not time out otherwise the end-user will assume that they have lost the connection to the servlet. A further description of this method and others that could be used is provided in a later section of this document.

The Q-Space in XML

Once the Q-PIT has completed its processing the Q-Space is sent to the servlet. The servlet needs to send a representation of this Q-Space to the end-user's device and in addition store the Q-Space so that it can be explored by the end-user. In effect the servlet is acting as a memory service for the device because, as we have already pointed out, a mobile phone or PDA cannot be expected to store and visualise a complete Q-Space. The XML document must capture all the information about a Q-Space, so as not to have to ask the Q-PIT for further

details. These details include the *schema* of the data source (the structure of each record), the records themselves and which records are in which groups.

Figure 6 shows how an attribute of the total schema is represented in XML. The servlet must request the schema from the Q-PIT. The Q-PIT returns a list of the attributes, such as ISBN, classmark, author, etc., which constitute the schema. Figure 6a shows the servlet request and one attribute that the Q-PIT returns, in this case the classmark attribute. Contained in the attribute information are the parameters that were used to create the Q-Space, together with the name and type. Figure 6b shows the corresponding representation in XML.

```
Servlet:    schema source=opac
Q-PIT:      ...
            source=opac number=0
            attribute=classmark type=text
            factor=1.0 exactness=0.0
            ...
```

Figure 6a: Part of a Q-PIT *schema* message.

```
<schema>
...
<attribute schema-id="0" type="text" factor="1.0"
exactness="0.0">
classmark
</attribute>
...
</schema>
```

Figure 6b: The same information in part of the XML document.

The representation of a search item is shown in figure 7b. Once the Q-Space has been generated the Q-PIT returns a list of search items; one of them is shown in figure 7a. The servlet reads each record and writes it into the XML document, shown in 7b. Notice how the attribute names are not written in the XML record but reference numbers instead. These numbers reference schema attribute records, like the one shown in figure 6b.

```
Q-PIT:      newobject query_id=555 id=5 ...
            data=isbn=1565923715&title=Java+examples+in+a+nutshell+%3A+a+tutorial+companio
            n+to+Java+in+a+nutshell.&author=FLANAGAN%2C+David&pub=O+Reilly&pubdate=19
            97&pubtype=Book&classmark=AZKF.J3
```

Figure 7a: A search item sent from the Q-PIT.

```
<results>
...
<result result-id="5">
<result-attribute reference-id="0">AZKF.J3</result-attribute>
<result-attribute reference-id="1">Java examples in a nutshell : a tutorial companion to Java in a
nutshell.</result-attribute>
<result-attribute reference-id="2">FLANAGAN, David</result-attribute>
<result-attribute reference-id="3">O Reilly</result-attribute>
<result-attribute reference-id="4">1997</result-attribute>
<result-attribute reference-id="5">Book</result-attribute>
<result-attribute reference-id="6">1565923715</result-attribute>
</result>
...
</results>
```

Figure 7b: The same search item represented in XML.

Finally the XML document contains the region information that the Q-PIT has generated. Figure 8a shows how the Q-PIT returns this information to the servlet, figure 8b shows the corresponding representation in XML. The reference numbers point to record numbers elsewhere in the XML document (for example: the record number of figure 7b was 5).

```
Q-PIT:    newGroup query_id=555 id=219 contains=3
          group query_id=555 id=219 object_id=62
          group query_id=555 id=219 object_id=63
          group query_id=555 id=219 object_id=60
```

Figure 8a: Part of a Q-PIT grouping item message.

```
</regions>
...
<region id="219">
<member reference="62"/>
<member reference="63"/>
<member reference="60"/>
</region>
...
</regions>
```

Figure 8b: The same information in part of the XML document.

Figure 9 shows the overall structure of the servlet-generated XML document. The schema, results and regions tags contain multiple instances of the XML content we have just shown. Once the servlet has this document the Cocoon publishing system can apply the custom WML stylesheet to it.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="wml.xsl" type="text/xsl" media="wap"?>
<?cocoon-process type="xslt"?>
<qspace>
   <datasource>OPAC</datasource>
   <keyword date="05/07/2000" time="18:32">java</keyword>
   <schema>
   ...
   </schema>
   <results>
   ...
   </results>
   <regions>
   ...
   </regions>
</qspace>
```

Figure 9: The overall structure of the XML document.

Displaying the Results

The first card that the user sees on their device, after having completed the sequence of processing decks, is shown in figure 10a. The information on the screen is calculated by the stylesheet itself – nowhere in the XML document does it actually specify the number of results returned. This number is calculated by the

stylesheet by counting the number of <record> instances within the document. The same situation is true for the group (region) information.

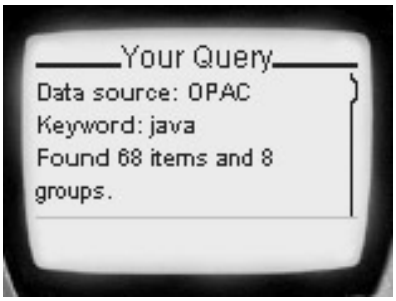


Figure 10a: Overview of the query

Figure 10b: Q-Space Options

Scrolling down to the base the card, the end-user can see the options that they have to view the Q-Space. The option selected in figure 10b is 'Browse Records'. This option permits simple sequential access to the search items, much like web page results from most common search engines, but with the addition of a hyperlink to the group that the search item is a member of. Figure 11a shows a typical search item from a Q-Space. Each attribute value is preceded by its attribute name with each name:value pair being on a separate line. To navigate onto the next record in the sequence the user can press the left-hand soft key, which is labelled as 'Next' in figure 11a. To return to the previous card that was viewed the user can press the right-hand soft key, labelled 'Back'. The card shown in figure 11a is larger than the device screen that it is displayed on, so the user has to scroll down the card to see all of the record attributes. Compare this to a web page of search items, displayed fifty at a time or a 3D Q-Space where every search item is visible!

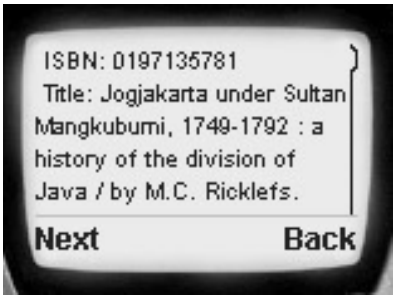

ISBN: 0197135781
Title: Jogjakarta under Sultan Mangkubumi, 1749-1792 : a history of the division of Java / by M.C. Ricklefs.
Next
Back

Figure 11a: The top of a typical record

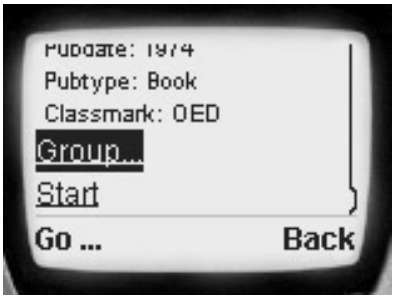


Figure 11b: The bottom of a typical record

When a user reaches the bottom of the card there are either one or two selectable items that can be clicked on. Figure 11b shows the base of a card with two selectable items. The first is a hyperlink to a description of the group that contains the currently viewed record; the second is a link back to the 'start' card, which contains the overall description of the Q-Space. All of the cards that contain record information have their own deck because we cannot rely on the end-user's device having enough capacity to hold all of the records in one deck.

The next option on the start card is 'Browse Groups' hyperlink, shown in figure 12a. Clicking on that hyperlink takes you to a list of the groups (or regions) in the Q-Space as shown in figure 12b. The number of items in each group is displayed next to the hyperlink that takes the user to more details about that particular group.

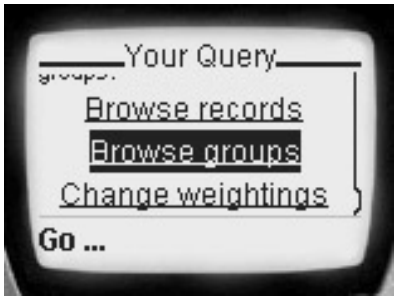


Figure 12a: Selecting group browsing

Figure 12b: A list of groups

When a user has chosen the group they want to explore, the servlet sends a new card containing a list of items in that group. Rather than showing every attribute value for each record in the group we have decided to show only one attribute, in the case of figure 13a this is the ISBN attribute. This view can be changed by the end-user if they select the 'Change View' hyperlink at the bottom of the item list. In response to this selection a new card is displayed containing a list of the attributes (figure 13b), which has been obtained from the schema section of the XML document (figure 6b). In figure 13b the user is selecting 'Author' as the attribute to list the items in the group by. Once the selection has been send to the servlet the stylesheet is told to create a new visualisation of the group based on the user's chosen attribute. The results can be seen in figure 13c.

Figure 13a: A group viewed by 'ISBN'

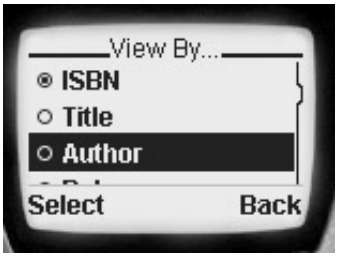


Figure 13b: Changing the group view

Figure 13c: A group viewed by 'Author'

When an end-user selects the hyperlink of an individual item a card containing the details of that item is loaded. It looks similar to the card shown in figure 11a, but instead of being able to navigate back and forth across the whole set of items, navigation is restricted to only those items in the group. This makes sure that the user cannot jump out of the 'group view' context, they have to explicitly choose to go back to record browsing if they want to see the whole set of items.

Changing Q-Space Parameters

When a user wants to change the way the Q-Space is generated they choose the 'Change Weightings' hyperlink at the bottom of the 'start' card (shown in figure 10b). In response to the request the servlet returns a deck of cards for changing the Q-Space parameter values. Figure 14a shows a table of attributes and the values associated with them, which is the initial card in this new deck. The table of values allows the user a complete overview of the every attribute's parameter values (bounded by the size of the display, of course). Each attribute in the table is selectable, and when an attribute is chosen, such as the 'ISBN' attribute in figure 14a, a card in the deck associated with that attribute is displayed (figure 14b).

Figure 14a: A table of attributes and their values

Figure 14b: Changing values for the ISBN attribute

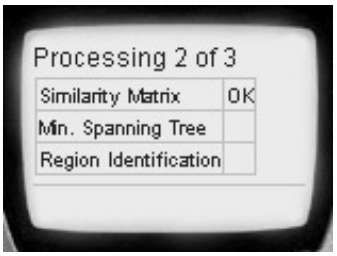


Figure 14c: Re-calculating the space

Using the input items on this card a user can modify the weighting or exactness parameters that affect the similarity calculations in the Q-Space (for an explanation of their actual role in the process see chapter 10). Once the user has set the values for that attribute the 'Change' hyperlink is selected and invoked. This returns to the card shown in figure 14a but with the new values shown next to the attribute that has been modified. All the new parameter values have to be held in memory on the phone itself because the XML on the servlet cannot be rewritten. Nor should it be, otherwise the Q-Space state that it represents becomes invalid. It is a representation of the Q-Space, not a writable storage space for temporary parameter values.

Once the user is happy with all the parameter values they can choose to regenerate the Q-Space based on them. The progress of this regeneration is shown in figure 14c. Note that there are only three functions being performed by the Q-PIT now. The 'Benediktine Domains' function, shown in figure 5, does not need to be carried out by the Q-PIT because the search items are still the same. Once this process has completed the user is shown a new 'start' card, similar to that shown in figure 10a, but with a different number of groups, and the Q-Space navigation process can begin again.

Implementation Issues

The biggest problem that was encountered in the implementation of this system was bridging the gap between the stateless and connectionless features of HTTP and the stateful and connected features of the Q-Space API. The servlet running within the Cocoon publishing system is at the boundary of these two communication techniques and is where all of the development took place to solve this issue. A servlet is a piece of Java code that is invoked by a 'server runner' in response to an HTTP request from a client. The servlet has access to all parts of the HTTP request that is being used to invoke it, allowing the servlet to tailor its behaviour. In our implementation the servlet responds to variables that are sent in an HTTP request (figure 4) by communicating with the Q-PIT that it has a connection to. In other situations a servlet may change the format of the content that it sends to a user based on the user-agent type specified in the HTTP request. In our case this is not important for the servlet because it is generating XML content. The user-agent type only becomes significant when choosing the stylesheet to apply to the XML document.

Sessions

The need for some sort of session management within the servlet is crucial because the end-user's device does not have a permanent connection to the servlet. It is also important to be able to distinguish between two devices that are accessing the servlet concurrently; otherwise one device could receive content that is meant for another. Fortunately this problem is also true for web-browsers, which also use the HTTP as a transport protocol. Java Servlets, which were designed with HTTP/HTML content in mind, have been created with built-in session management. Servlet technology provides several mechanisms for session tracking. The first uses cookies[1], a small text file that is stored on the client side and can be queried and modified by the server that has issued them. Unfortunately we cannot rely on end-user WAP devices to support the Cookies protocol or even waste precious resources being able to store them. A further session tracking method that servlets provide is the re-writing of URLs, where every URL that is presented to the user has been dynamically modified to include a unique session identifier. Although HTTP is a stateless protocol through the use of URL re-writing we can provide state and session features; a HTTP request contains both a session identifier and additional parameters that indicate the current state that the user believes the session to be in.

1 "HTTP State Management Mechanism", D. Kristol and L. Montulli, February 1997. URI: http://www.ietf.org/rfc/rfc2109.txt

Connection versus Connectionless

As we have pointed out earlier in this document, the Q-PIT to servlet connection is open throughout the lifetime of the Q-Space but the phone/servlet connection is connectionless. Compounding this problem is that the initiation of the connection is only one way; the phone connects to the servlet as it needs to where as the servlet has no way of initiating a connection to the phone. This problem is particularly acute when trying to show the progress of the Q-PIT calculations. When using HTML there are several ways of tackling this problem but unfortunately most of them do not transfer to WML. For example, an HTML page can begin to be rendered by a web browser before the entire page has been received. We could write the Q-PIT progress messages directly to a page or deck being loaded by the client but in the case of WML, which is based in XML, the content can be validated. This means that the client browser will check to see that the WML to be displayed is well formed, that all WML tags have been opened, nested and closed correctly. This validation cannot take place whilst there is more of the WML to load so the content cannot be displayed. We cannot display the progress of the Q-PIT calculations directly using this method because the WML browser would only display the card once all of the content has been sent – defeating the whole idea of the progress messages, a near real-time indication of the Q-PIT's state. We could have used an animated image to indicate *activity* rather than progress, similar to the animated icons in the top right corner of most common web browsers, but in terms of the time taken to generate larger Q-Spaces, this type of feedback is not really appropriate. It gives the user no indication of what has been done what is actually happening and when it can be expected to finish. Scripting on the client side might have been an option; a deck containing pre-authored status messages could be sent to the phone and the swapping of cards could have been controlled by the servlet. This idea would provide some sort of real-time status information but, as we have already pointed out, the servlet has no way of telling the phone to show the next card in the sequence because of the client-pull nature of the connection.

Figure 5 shows a card from the sequence indicating progress through Q-Space generation process. Each step in the creation of a Q-Space is shown in a table, when a step has been completed an 'OK' is placed in the correct position in the table. This method informs the user as to which step the Q-PIT is currently doing, it does not indicate progress within a step (e.g. progress through the intensive similarity calculation). A finer granularity of progress information would mean an increase in the messages, decks and cards that would have to be sent to a end-user's phone. In situations where a mobile network provider charges for the amount of content that is sent to the phone it is desirable to keep the amount of progress information to a minimum. It is only transient information and has no impact on the generation of the Q-Space itself or its use. The solution that we have used (shown in figure 5) is to package the coarsely grained progress information into cards and separate them into individual decks. There is a script in

each card that, after a short timeout, will request the next deck/card in the progress sequence. We use the fact that the WML content will be validated before it is displayed and make the servlet return only the first few characters of the new WML deck. These describe the deck itself and not the content of the card. Once the servlet has received the progress message from the Q-PIT it can finish writing the WML deck and the end-user's phone can display it. This deliberate stalling of the WML response will work up to a certain point. If the stall is too long the WML browser, or the gateway that it is connecting to, may assume that the connection is dead and close the browser connection to the servlet. To get around this problem we need to select an appropriate 'wait time' that each card uses before sending a request for the next card in the sequence.

The technique that we have finally used is not particularly elegant and is purely to get around the restrictions imposed by the HTTP architecture. In the future, as end-user devices become more powerful, Java applets based on the Micro edition of Java[1] can be downloaded to the device and used to establish a connection to the servlet, or even directly to the Q-PIT. The use of applets in this way mirrors their use in HTML, for tasks such as live chat boards or online games.

Conclusion

The system described above has been a first attempt to visualise a Q-Space on a resource-limited device. The realisation of this system has exploited the relatively new XML and XML stylesheet technology, which has been implemented by the open source Cocoon server. This enforces the separation between presentation and content that was first proposed in the design of the Q-PIT itself, in chapter 6, and the Q-PIT engine, in chapter 10. During the implementation of the system the difference between the connection and connectionless protocols proved to be problematic but a solution was found that reverted to a page-based metaphor rather than the WML card-and-deck approach. A 'push' protocol has been defined for the WAP architecture[2] but this requires the server and gateway to support push protocol. The Cocoon server that we use is not a specialist WAP server but a generic HTTP server, primarily meant for standard client pull HTTP access.

By converting the Q-Space representation to WML through the use of XML stylesheets we have an effective way of persistently storing a Q-Space and decoupling its visualisation from the data it contains. The fact that Cocoon can differentiate between end-user agents, be they mobile phones, web browsers or 3D viewers, and select the appropriate stylesheet allows the reuse of all of the servlet/Q-PIT infrastructure. To add additional visualisations using different languages or file forms only requires the creation of a new stylesheet that can

1 http://www.java.sun.com/j2me/

2 http://www1.wapforum.org/tech/terms.asp?doc=SPEC-PushArchOverview-19991108.pdf

convert the Q-Space document into that language. We recommend that the future development of abstract landscapes, where there is no right or wrong way to represent an abstract artefact, should, wherever possible, describe this artefact a platform-neutral a manner as possible. XML and its stylesheet technology has been shown to be an acceptable solution to this requirement.

Concluding Remarks

Tom Rodden
Lancaster University

We wish conclude this deliverable with a few very brief remarks on the work presented in this deliverable. The three years of the project has seen the presentation of a diverse collection of activities that have drawn upon a broad set of supporting disciplines. Each of these activities have provided significant advances in terms of new understandings of electronic environments, new approaches to design and development, new supporting infrastructures and new interactive devices.

In addition to these different particular contributions the project has actively sought from the outset to support and promote integration and during the project considerable integration has taken place. In particular we have seen:

- The construction of a unified demonstrator that conveys the core aspects of the project and pulls together many of the different demonstrators realised during the work of the project.
- The agreement of a common methodological approach to the development of electronic landscapes and a convergence between the understanding of developers and multimedia artists.
- The construction of generic architectures that underpin different electronic environment and allow these to be accessed from different devices.
- The exploration of how people use and experience on-line electronic environments.

These different lessons have advanced our understanding not only of electronic environments realised using virtual reality techniques but have increasingly, throughout the project, embraced the situation where physical environments are augmented with interactive devices. For example, in the Placeworld demonstrator we provide an installation of this environment that is fundamentally based on the installation of specialised equipment within a physical setting.

The merge between the physical and the virtual is hardly surprising given the initial aims of the eSCAPE project to explore a space that brings other spaces together. Indeed, we would envisage that the fundamental lessons on structuring of the underlying architecture, the approach to the development of these environments and the construction of new forms of hybrid space will readily transform to the development of physical environments that embody interactive devices.

The final demonstrator Placeworld illustrates a significant infrastructure to allow a number of electronic environments to co-exist. The final developed demonstrator also provides a significant dissemination device to the general public. As an installation where the work of the project is on display, it illustrates a novel way of presenting and experiencing on-line electronic environments through novel interactive devices. However, the choice to realise the demonstrator on a platform that is open source and to consider the packaging of the device into a format that can be readily disseminated allows us to seed the eventual development of a broad electronic landscape. We are currently finalising the production of a final version of this demonstrator that can be used on normal personal computers.